Méthodes formelles pour le respect de la vie privée par construction

Thibaud Antignac

N° d'ordre 2015ISAL0016 Année 2015

Thèse

Méthodes formelles pour le respect de la vie privée par construction

Présentée devant
L'Institut National des Sciences Appliquées de Lyon

Pour obtenir
Le grade de docteur

Spécialité : Informatique
École doctorale : Informatique et Mathématiques (InfoMaths — EDA 512)

Par
Thibaud Antignac

Soutenance le 25 février 2015 devant la Commission d'examen

Jury

Rapporteurs	Sébastien Gambs	Chaire de recherche, HDR	Université de Rennes 1, Inria
	Gerardo Schneider	Professeur	Université de Göteborg
Examinateurs	Antonio Kung	Directeur technique	Trialog
	Patricia Serrano Alvarado	Maître de conférences	Université de Nantes
Président	Fabrice Valois	Professeur des universités	INSA de Lyon
Directeur	Daniel Le Métayer	Directeur de recherche	Inria

Laboratoire de recherche
Centre of Innovation in Telecommunications and Integration of service (CITI lab)
Inria, Université de Lyon/INSA de Lyon

Les travaux présentés dans cette thèse ont été réalisés sous la direction de Daniel Le Métayer au sein de l'équipe Privatics au laboratoire CITI (Inria, Université de Lyon/INSA de Lyon). Ceux-ci ont été financés par une subvention CoRDI-S Inria et par le projet européen PRIPARE (FP7-ITC-2013-1.5).

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

INSA Direction de la Recherche
Ecoles Doctorales - Quinquennal 2011-2015

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **http://www.edchimie-lyon.fr** Sec : Renée EL MELHEM Bat Blaise Pascal 3e etage 04 72 43 80 46 Insa : R. GOURDON	M. Jean Marc LANCELIN Université de Lyon – Collège Doctoral Bât ESCPE 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : 04.72.43 13 95 directeur@edchimie-lyon.fr
E.E.A.	**ELECTRONIQUE, ELECTROTECHNIQUE, AUTOMATIQUE** **http://edeea.ec-lyon.fr** Sec : M.C. HAVGOUDOUKIAN eea@ec-lyon.fr	M. Gérard SCORLETTI Ecole Centrale de Lyon 36 avenue Guy de Collongue 69134 ECULLY Tél : 04.72.18 60.97 Fax : 04 78 43 37 17 Gerard.scorletti@ec-lyon.fr
E2M2	**EVOLUTION, ECOSYSTEME, MICROBIOLOGIE, MODELISATION** **http://e2m2.universite-lyon.fr** Secrétariat : Safia AIT CHALAL Bât Atrium – Service des études doctorales – 1er étage Tél : 04 72 44 83 62 Fax : 04 72 43 13 06 Safia.ait-chalal@univ-lyon1.fr	Fabrice CORDEY Laboratoire de Géologie de Lyon Directeur Terre, Planète et Environnement Université Claude Bernard Lyon1 Bât. Géode – R2 – Bureau 225 43 bd du 11 novembre 1918 69622 VILLEURBANNE Cedex Tél : +33 (0)4 72 44 83 74 fabrice.cordey@univ-lyon1.fr Sylvie REVERCHON Microbiologie, Adaptation, Pathogénie Directeur adjoint UMR 5240 CNRS UCBL INSA BayerCropScience Représentant INSA Domaine Scientifique de la Doua, Université Lyon 1 Bat. André LWOFF, 10 rue Raphaël Dubois 69622 Villeurbanne cedex Tel +33 (0) 4 72 43 26 95 sylvie.reverchon-pescheux@insa-lyon.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTE** **http://www.ediss-lyon.fr** Secrétariat : Safia AIT CHALAL Bât Atrium – Service des études doctorales – 1er étage Tél : 04 72 44 83 62 Fax : 04 72 43 13 06 Safia.ait-chalal@univ-lyon1.fr	Mme Emmanuelle CANET-SOULAS INSERM U1060, CarMeN lab, Univ. Lyon 1 Bâtiment IMBL 11 avenue Jean Capelle INSA Lyon 69621 Villeurbanne Tél : 04.72.11.90.13 Emmanuelle.canet@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHEMATIQUES** **http://infomaths.univ-lyon1.fr** Sec :Renée EL MELHEM Bat Blaise Pascal 3e etage infomaths@univ-lyon1.fr	Mme Sylvie CALABRETTO LIRIS – INSA Lyon Bat Blaise Pascal 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72. 43. 80. 46 Fax 04 72 43 16 87 Sylvie.calabretto@insa-lyon.fr

Matériaux	**MATERIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry Ed.materiaux@insa-lyon.fr	M. Jean-Yves BUFFIERE INSA Lyon MATEIS Bâtiment Saint Exupéry 7 avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72.43 71.70 Fax 04 72 43 85 28 Ed.materiaux@insa-lyon.fr
MEGA	**MECANIQUE, ENERGETIQUE, GENIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec : M. LABOUNE PM : 71.70 –Fax : 87.12 Bat. Saint Exupéry mega@insa-lyon.fr	M. Philippe BOISSE INSA Lyon Laboratoire LAMCOS Bâtiment Jacquard 25 bis avenue Jean Capelle 69621 VILLEURBANNE Cedex Tél : 04.72 .43.71.70 Fax : 04 72 43 72 37 Philippe.boisse@insa-lyon.fr
ScSo	**ScSo*** **http://recherche.univ-lyon2.fr/scso/** Sec : Viviane POLSINELLI Brigitte DUBOIS Insa : J.Y. TOUSSAINT viviane.polsinelli@univ-lyon2.fr	Mme Isabelle VON BUELTZINGLOEWEN Université Lyon 2 86 rue Pasteur 69365 LYON Cedex 07 Tél : 04 78 69 72 76

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Remerciements

Les remerciements sont souvent la dernière étape de la rédaction d'un manuscrit de thèse mais rappellent les souvenirs du début.

À cette occasion, je tiens à remercier en premier lieu mon directeur de thèse, Daniel Le Métayer. L'accueil qu'il m'a réservé au sein de l'équipe m'a permis de commencer ce travail doctoral dans des conditions idéales. Ses conseils, sa confiance, voire même parfois sa patience et son dévouement pendant ces travaux m'ont aidé à acquérir la démarche scientifique nécessaire à son aboutissement. La liberté qu'il m'a laissée m'a permis de prendre plaisir à défricher les sentiers du respect de la vie privée.

Un grand merci aux rapporteurs, Sébastien Gambs et Gerardo Schneider, pour avoir accepté de rapporter cette thèse et pour leur déplacement pour la soutenance. Je remercie aussi les autres membres du jury, Antonio Kung, Patricia Serrano Alvarado et Fabrice Valois, pour avoir aussi accepté d'examiner ce travail et de venir à Lyon.

Ce travail n'aurait pas pu se dérouler sans l'accueil des directeurs du centre Inria de Grenoble - Rhône-Alpes et de CITI lab qu'étaient François Sillion et Jean-Marie Gorce au moment où j'ai commencé ma thèse. Les autres membres, anciens et nouveaux, de l'équipe Privatics, lyonnais comme grenoblois, ont aussi joué un rôle important à travers les discussions parfois-mais-pas-toujours scientifiques que nous avons pu avoir.

Plus largement, je souhaite exprimer ma gratitude envers l'ensemble des membres du laboratoire, sans en citer de peur d'en oublier, pour la bonne humeur et la sympathie qui font du laboratoire un endroit de vie scientifique agréable et débordant d'émulation. Un merci particulier aussi envers les assistantes d'équipe, Gaëlle, Helen et Joëlle, pour leur aide, leur soutien et leur bonne humeur.

Aussi je tiens à remercier mes amis, ma famille et ma belle-famille pour leur écoute et leur compréhension durant ces années. J'ai apprécié les bons moments passés ensemble et attends avec hâte les prochains.

Enfin, ces remerciements seraient corrects mais incomplets sans un énorme merci à Thomas qui a été d'un soutien sans faille en toutes circonstances et n'a cessé de m'encourager pendant toutes ces années depuis Bordeaux, Orléans ou Paris.

Résumé

Le respect de la vie privée par construction est de plus en plus mentionné comme une étape essentielle vers une meilleure protection de la vie privée. Les nouvelles technologies de l'information et de la communication donnent naissance à de nouveaux modèles d'affaires et de services. Ces services reposent souvent sur l'exploitation de données personnelles à des fins de personnalisation. Alors que les exigences de respect de la vie privée sont de plus en plus sous tension, il apparaît que les technologies elles-mêmes devraient être utilisées pour proposer des solutions davantage satisfaisantes. Les technologies améliorant le respect de la vie privée ont fait l'objet de recherches approfondies et diverses techniques ont été développées telles que des anonymiseurs ou des mécanismes de chiffrement évolués.

Cependant, le respect de la vie privée par construction va plus loin que les technologies améliorant simplement son respect. En effet, les exigences en terme de protection des données à caractère personnel doivent être prises en compte au plus tôt lors du développement d'un système car elles peuvent avoir un impact important sur l'ensemble de l'architecture de la solution. Cette approche peut donc être résumée comme « prévenir plutôt que guérir ».

Des principes généraux ont été proposés pour définir des critères réglementaires de respect de la vie privée. Ils impliquent des notions telles que la minimisation des données, le contrôle par le sujet des données personnelles, la transparence des traitements ou encore la redevabilité. Ces principes ne sont cependant pas suffisamment précis pour être directement traduits en fonctionnalités techniques. De plus, aucune méthode n'a été proposée jusqu'ici pour aider à la conception et à la vérification de systèmes respectueux de la vie privée.

Cette thèse propose une démarche de spécification, de conception et de vérification au niveau architectural. Cette démarche aide les concepteurs à explorer l'espace de conception d'un système de manière systématique. Elle est complétée par un cadre formel prenant en compte les exigences de confidentialité et d'intégrité des données. Enfin, un outil d'aide à la conception permet aux concepteurs non-experts de vérifier formellement les architectures. Une étude de cas illustre l'ensemble de la démarche et montre comment ces différentes contributions se complètent pour être utilisées en pratique.

Mots-clés : respect de la vie privée, méthodes formelles, vérification, logiques, génie logiciel, conception, architecture

Abstract

Privacy by Design (PbD) is increasingly praised as a key approach to improving privacy protection. New information and communication technologies give rise to new business models and services. These services often rely on the exploitation of personal data for the purpose of customization. While privacy is more and more at risk, the growing view is that technologies themselves should be used to propose more privacy-friendly solutions. Privacy Enhancing Technologies (PETs) have been extensively studied, and many techniques have been proposed such as anonymizers or encryption mechanisms.

However, PbD goes beyond the use of PETs. Indeed, the privacy requirements of a system should be taken into account from the early stages of the design because they can have a large impact on the overall architecture of the solution. The PbD approach can be summed up as "prevent rather than cure".

A number of principles related to the protection of personal data and privacy have been enshrined in law and soft regulations. They involve notions such as data minimization, control of personal data by the subject, transparency of the data processing, or accountability. However, it is not clear how to translate these principles into technical features, and no method exists so far to support the design and verification of privacy compliant systems.

This thesis proposes a systematic process to specify, design, and verify system architectures. This process helps designers to explore the design space in a systematic way. It is complemented by a formal framework in which confidentiality and integrity requirements can be expressed. Finally, a computer-aided engineering tool enables non-expert designers to perform formal verifications of the architectures. A case study illustrates the whole approach showing how these contributions complement each other and can be used in practice.

Keywords: privacy, formal methods, verification, logics, software engineering, design, architecture

Table des matières

Liste des tableaux

Liste des figures

Liste des algorithmes

CHAPITRE 1

Introduction

1.1 Contexte

Le respect de la vie privée par construction (*PbD* dans la suite) [1] est souvent désigné comme une approche nécessaire pour améliorer le respect de la vie privée [2] dans la société numérique. Il pourrait même devenir une obligation légale si le projet de règlement européen pour la protection des données personnelles (*GDPR* dans la suite) [3] [GDPR14] était adopté par le Conseil de l'Union européenne après avoir obtenu le vote du Parlement européen au mois de mars 2014.

Le fait que de futurs cadres légaux promeuvent ou imposent le *PbD* est une étape significative mais son adoption prendra du temps. De plus, les interprétations des principes relatifs au respect de la vie privée peuvent eux-même varier de manière importante. En effet, la notion de vie privée est composite et évolue dans le temps et selon les pays, ce qui exclut toute définition précise dans un document légal à large portée.

Par conséquent, l'adoption de textes législatifs incorporant le *PbD* ne doit pas être vue comme un aboutissement : la question de son adoption réelle reste entière. En effet, la situation politique décrite dans la Section 1.1.1 et les tensions économiques présentées dans la Section 1.1.2 ont autant d'importance que le cadre juridique qui fait l'objet de la Section 1.1.3.

1.1.1 Situation politique

Le respect de la vie privée est une notion complexe et très dépendante du contexte social (géographique, politique, . . .). Aborder ce sujet demande une mise en perspective politique tant cette notion est à la fois multiforme et sensible.

1. Diverses traductions existent pour le terme anglais de *Privacy by Design* : respect de la vie privée dès, ou par, la conception, ou la construction, sont les plus fréquemment rencontrées. Nous retenons l'expression « par construction » car il existe des phases précédant la conception proprement dite dans lesquelles cette approche s'applique.
2. Le sens que nous retiendrons est celui rattaché au terme anglais de *privacy*. Les moyens informatiques pour la protéger relèvent de la protection des données.
3. *General Data Protection Regulation* en anglais.

Le respect de la vie privée a pris de plus en plus de place dans l'actualité récente suite à de nombreuses affaires qui ont culminé avec les révélations de l'affaire dite Snowden, du nom de l'ex-analyste ayant travaillé pour la *CIA* [4]. Celui-ci a révélé plusieurs jeux de documents démontrant des activités massives de surveillance de la part de cette agence et de ses homologues dans des pays étrangers [Gre14]. À la fois la qualité et la quantité des données concernées ont fait apparaître la question du respect de la vie privée des citoyens à l'ordre du jour des plus importantes réunions internationales.

Ces activités de surveillance massive portent préjudice aux citoyens, ressortissants ou non des pays concernés. Ces derniers font l'objet d'intrusions dans leur vie privée. Ces intrusions sont encadrées par des droits d'exception et des lois parfois secrètes que les États justifient par les besoins de sûreté du territoire, de sécurité des personnes et de lutte contre le terrorisme.

Les citoyens ne sont cependant pas les seules victimes de cette exploitation à grande échelle. Les États eux-mêmes le sont sur le plan fiscal. Des initiatives parlementaires considèrent que la fourniture de ces données personnelles dans le cadre de services commerciaux constitue un échange de valeur entre l'utilisateur et le fournisseur du service. Comme tout échange, celui-ci devrait pouvoir être taxé. Des propositions ont été faites dans ce sens par des commissions législatives sans déboucher pour le moment sur des mesures contraignantes [CC13].

L'importance du respect de la vie privée ne relève donc pas seulement de la relation entre citoyens et États. Il est aussi en lien avec l'utilisation économique de plus en plus massive des données personnelles par les industries et services numériques.

1.1.2 Pression économique

Le numérique s'impose dans toutes les filières industrielles et de services et remplace progressivement les acteurs traditionnels [And11] qui n'ont pas pu ou pas voulu s'adapter à temps, dans la ligne du principe de la destruction créatrice [Sch42]. Ce succès s'explique en partie par l'exploitation fine, intensive [5] et permise à grande échelle [6] par les traitements automatisés [Man+11]. Ceux-ci permettent des services de plus en plus personnalisés et promettent de prévoir et de satisfaire bientôt les désirs des individus avant même que ces derniers n'en soient conscients [7].

Pour y parvenir, les utilisateurs sont invités par les fournisseurs de services à partager la plus grande quantité possible de données pour des exploitations directes (rendre un meilleur service) ou indirectes (afficher des publicités ciblées par exemple). Cette accumulation centralisée de données comporte des risques importants en terme de fuites d'information ou d'utilisations indues [Sol06]. Celles-ci peuvent avoir des impacts importants sur les utilisateurs qui peuvent être profilés et discriminés [Odl03] ou encore voir leurs données rendues publiques [HG08].

Les enquêtes d'opinion montrent que les utilisateurs sont attentifs à la notion de respect de la vie privée [ITS14]. Des expériences ont toutefois montré qu'il n'était pas évident que les utilisateurs

4. *Central Intelligence Agency*, agence de renseignement des États-Unis.
5. *Data science* en anglais.
6. *Big data* en anglais.
7. Voir par exemple les services Google Now (http://www.google.com/landing/now/) ou Amazon Echo (http://www.amazon.com/echo/).

soient prêts à payer plus cher pour un service plus respectueux de leur vie privée [BKP12]. Tout se passe comme si le respect de la vie privée était appréciable sans toutefois que son absence soit rédhibitoire pour les utilisateurs.

L'utilisation des données personnelles par les acteurs économiques permet l'émergence de nouveaux services personnalisés à forte valeur ajoutée. Cette utilisation crée une tension entre les dangers d'une mauvaise utilisation et les opportunités qu'elle représente. En réaction à ce phénomène, les législateurs et organisations publiques ont dû se prononcer à de multiples reprises pour proposer des moyens juridiques qui suivent et encadrent l'évolution des usages.

1.1.3 Cadre juridique

La notion de vie privée n'est pas récente en matière juridique. Elle apparaît dans le droit dès la fin du XIXe siècle avec l'avènement de la photographie [WB90]. Son respect est alors défini comme « le droit d'être laissé seul » [8]. En Europe, elle a été consacrée par l'article 8 de la CDHL [9] qui dit que « [t]oute personne a droit au respect de sa vie privée » [Con50]. Depuis, une autre vision a émergé : celle de vie privée comme droit à l'auto-détermination informationnelle [Wes70] selon l'expression [10] retenue par la cour constitutionnelle fédérale allemande en 1983 [GFC83].

L'appréciation de la valeur individuelle ou collective de la vie privée donne également lieu à différentes approches. La première, plutôt d'inspiration anglo-saxonne, tend à considérer les données personnelles comme un bien qui peut être vendu ou échangé en laissant les citoyens « choisir leur niveau optimal de respect de la vie privée sans intervention parentale de l'État » [11] [Pos73]. Au contraire, l'approche européenne privilégie la protection de la vie privée comme bien à la fois personnel et collectif tant elle participe à la construction de la société démocratique [RP09].

En pratique, deux types majeurs de textes juridiques officiels peuvent être distingués : les textes non contraignants et ceux qui le sont. Pour ce qui est de la première catégorie, plusieurs organisations ou agences internationales ont émis des positions et des recommandations sur un traitement des données personnelles respectueux de la vie privée. Dès 1980, l'OCDE [12] a proposé des lignes de conduite[OEC80]. Celles-ci recommandent notamment de limiter la collecte de données personnelles, d'assurer leur qualité et de circonscrire leur usage à des finalités bien spécifiées. Leur collecte doit aussi être limitée à ce qui est nécessaire pour réaliser ces finalités. Enfin, celui qui a collecté les données a la responsabilité de les conserver de manière sécurisée et de permettre aux personnes concernées de les modifier. Ces principales notions ont été reprises dans d'autres propositions telles que celles de la *FTC* [13] [RCRC73]

8. « [*T*]*he right to be let alone* » dans le texte.
9. Convention de sauvegarde des Droits de l'Homme et des Libertés fondamentales.
10. *Informationelle Selbstbestimmung* en allemand.
11. « [*C*]*hoose their optimal mix of privacy without parental intervention from the State* » dans le texte.
12. Organisation de Coopération et de Développement Économiques, *Organisation for Economic Co-operation and Development* en anglais.
13. *Federal Trade Commission*, agence de régulation de la consommation et de la concurrence des États-Unis.

ou du groupe de travail dit « G29 » [14] [FoP09] qui rassemble les autorités de protection des données personnelles des pays membres de l'Union Européenne.

Parmi les textes contraignants apparaissent notamment les lois, directives et règlements. Beaucoup de pays ont légiféré sur les données personnelles, y compris des entités supranationales telles que la Commission Européenne qui a émis une directive régissant l'utilisation des données personnelles [Dir95]. Des règlementations sectorielles [15] ont aussi été proposées pour cibler des activités sensibles [Dir95 ; HIPPA96]. Cependant, le texte qui pourrait avoir le plus d'influence dans le futur proche, tout du moins en Europe, est le projet de *GDPR* [GDPR14], qui a été voté par le Parlement européen au mois de mars 2014, s'il est aussi adopté par le Conseil de l'Union européenne. Ce texte définit clairement les acteurs en jeu dans son article 4 :

- le **sujet** qui est la personne physique à qui se rapportent les données personnelles : celles-ci peuvent être identifiées ou identifiables directement ou indirectement à travers des moyens raisonnables par toute autre personne [16] ;
- le **responsable de traitement** qui est la personne physique ou morale qui détermine les finalités, conditions et moyens de traitement des données personnelles [17] ;
- l'**opérateur** qui est la personne physique ou morale qui effectue le traitement des données personnelles au nom du responsable de traitement [18].

Ces définitions reposent sur celle du **traitement** qui est l'opération ou l'ensemble des opérations effectuées sur des données personnelles par des moyens automatisés ou non [19].

Le règlement dispose en son article 5 que doivent être respectés, entre autres, les principes de :

- la **limitation de la finalité** qui soumet la collecte des données à des finalités spécifiques, explicites et légitimes [20] ;
- la **minimisation des données** qui limite la collecte et le traitement des données au minimum nécessaire pour atteindre la finalité [21] ;
- la **précision** qui requiert que les données personnelles soient précises, maintenues à jour et rectifiées en cas de besoin [22].

14. *Working Party 29* en anglais, groupe de travail composé des vingt-neuf autorités de protection des données personnelles des pays membres de l'Union européenne.

15. Moyen privilégié de règlementation aux États-Unis dans ce contexte.

16. « [*I*]*dentified natural person or* [. . .] *natural person who can be identified, directly or indirectly, by means reasonably likely to be used by the controller or by any other natural or legal person, in particular by reference to an identification number, location data, online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that person* » dans le texte.

17. « [*N*] *atural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data* » dans le texte.

18. « [*N*]*atural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller* » dans le texte.

19. « [*O*]*peration or set of operations which is performed upon personal data or sets of personal data, whether or not by automated means, such as collection, recording, organization, structuring, storage, adaptation or alteration, retrieval, consultation, use, disclosure by transmission, dissemination or otherwise making available, alignment or combination, erasure or destruction* » dans le texte.

20. « [*Personal data shall be*] *collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes (purpose limitation)* » dans le texte.

21. « [*Personal data shall be*] *adequate, relevant, and limited to the minimum necessary in relation to the purposes for which they are processed ; they shall only be processed if, and as long as, the purposes could not be fulfilled by processing information that does not involve personal data (data minimisation)* » dans le texte.

22. « [*Personal data shall be*] *accurate and, where necessary, kept up to date ; every reasonable step must be*

Enfin, ce projet de règlement mentionne explicitement dans son article 23 le *PbD* [23] qui devra être adopté par les responsables de traitement. Dans ce cadre, la protection des données à caractère personnel devra être assurée tout au long de « la gestion [de leur] cycle de vie » [24]. Une évaluation de l'impact sur la vie privée [25] couvrant elle aussi toutes les étapes de « la gestion du cycle de vie des données personnelles » [26] est aussi imposée par l'article 33.

Ces contraintes de plus en plus fortes sur les industriels et fournisseurs de services amènent ceux-ci à devoir changer leur pratiques. Des moyens techniques doivent donc être créés et mis en œuvre pour permettre une démarche adoptant le *PbD* [LM10].

1.2 Besoins et contributions

Bien que les dimensions politiques, économiques et juridiques soient essentielles et que toutes les pistes possibles pour favoriser l'adoption du *PbD* doivent être explorées, nous nous concentrons dans la suite de ce travail sur son volet technique.

De plus en plus de technologies améliorant le respect de la vie privée [27] sont aujourd'hui disponibles pour fournir des garanties importantes dans des contextes et applications diverses. Même si les recherches sur ces technologies doivent être poursuivies (et amplifiées), une des questions majeures qui se pose désormais est la bonne utilisation des techniques qui sont déjà disponibles.

Cette thèse est composée de trois parties qui répondent à ce besoin par :

1. **Une démarche de génie logiciel mettant en œuvre le *PbD*.** La plupart des travaux existant dans le domaine du respect de la vie privée se concentrent sur les technologies et les briques de base des systèmes. L'impact du *PbD* sur le cycle de conception est plus rarement abordé. Nous proposons une démarche systématique permettant à un concepteur non-expert de concevoir une architecture et de la vérifier par rapport à ses exigences.
2. **Un modèle formel intégré à la démarche de *PbD*.** La vie privée est une notion intrinsèquement complexe qui peut, ou semble parfois, entrer en conflit avec d'autres exigences. Elle doit donc être définie de manière précise et non ambiguë. Les modèles formels sont particulièrement adaptés pour atteindre cet objectif. Ils rendent possible la définition précise des concepts manipulés et permettent de vérifier les propriétés d'une solution, de raisonner sur les choix architecturaux et de justifier ces derniers (contribuant ainsi

taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay (accuracy) » dans le texte.

23. En utilisant le terme de *data protection* plutôt que celui de *privacy*.

24. « *Data protection by design shall have particular regard to the entire lifecycle management of personal data from collection to processing to deletion, systematically focusing on comprehensive procedural safeguards regarding the accuracy, confidentiality, integrity, physical security and deletion of personal data* [*in particular with regard to the principles laid out in Article 5.*] » dans le texte.

25. *Privacy impact assessment (PIA)* en anglais.

26. « *The assessment shall have regard to the entire lifecycle management of personal data from collection to processing to deletion.* » dans le texte.

27. *Privacy-Enhancing Technologies (PETs)* en anglais.

également à la notion de redevabilité [28]). Nous proposons un cadre formel remplissant ces conditions et compatible avec la démarche proposée précédemment.

3. **Un outil d'aide à la conception et à la vérification, *CAPRIV*, implémentant la démarche et le modèle de *PbD*.** Les concepteurs ne sont généralement des experts ni en *PbD*, ni en méthodes formelles. Ils ont donc besoin d'environnements dédiés basés sur des interfaces aisées à prendre en main. Celles-ci doivent les guider dans la démarche de conception et les préserver des détails mathématiques des modèles formels. Nous avons conçu *CAPRIV* [29], un outil d'aide à la conception et à la vérification qui permet au concepteur de spécifier ses exigences, de concevoir une architecture selon la démarche présentée et de vérifier formellement ses propriétés.

Ces trois contributions [ALM14b] sont détaillées successivement dans les Chapitres 3, 4 et 5. Elles sont précédées d'un état de l'art dans le Chapitre 2. Le Chapitre 6 présente une étude de cas avant le Chapitre 7 qui conclut le document et dresse des perspectives de recherche futures.

28. *Accountability* en anglais, parfois traduit par « responsabilité ».
29. Pour *Computer-Aided Privacy engineering tool*, outil d'aide à la conception respectueuse de la vie privée.

CHAPITRE 2

État de l'art

Le domaine d'application de cette thèse est le respect de la vie privée dans des systèmes d'information manipulant des données personnelles. La réalisation de tels systèmes fait généralement appel à des outils techniques pour la manipulation et la protection de ces données. Leur développement suit un cycle faisant intervenir plusieurs acteurs qui aboutit à une solution devant satisfaire toutes les exigences qui pèsent sur le système. Ces méthodes du génie logiciel peuvent être appliquées à cet effet. Quand des garanties plus fortes sont exigées, les méthodes formelles peuvent être utilisées pour vérifier que le système conçu satisfait bien ces exigences. Les méthodes formelles peuvent aussi fournir une aide dans la phase de conception du système.

Cette thèse se situe donc au carrefour de trois domaines scientifiques mis au service du respect de la vie privée qui sont les outils techniques s'y rapportant en Section 2.1, le génie logiciel en Section 2.2 et les méthodes formelles en Section 2.3. Enfin, une caractérisation de différentes solutions issues de la littérature pour le domaine du relevé intelligent de consommation électrique est proposée dans la Section 2.4.

2.1 Outils techniques

Les outils techniques utilisés ont un impact majeur sur le respect de la vie privée de leurs utilisateurs et de la règlementation. De nouveaux critères d'analyse ont dû être créés pour qualifier et quantifier le degré de protection offert par les techniques de protection de la vie privée. Nous en fournissons un aperçu dans la Section 2.1.1. Les techniques de protection disponibles, qui sont décrites dans la Section 2.1.2, reposent pour la plupart sur des schémas cryptographiques. Ces primitives cryptographiques sont décrites dans la Section 2.1.3.

2.1.1 Critères de respect de la vie privée

Les critères les plus couramment utilisés dans le domaine proviennent des communautés scientifiques s'intéressant à la sécurité et à la cryptographie d'une part et aux bases de données d'autre part. La première s'attache surtout à la confidentialité des données et à la protection de

l'identité des sujets et des événements d'un système alors que la seconde se concentre sur les modes de stockage et de restriction d'accès aux données et à leur usage.

Dans la première catégorie, les notions les plus souvent utilisées sont les suivantes [PH10] :

- l'**anonymat** qui caractérise le fait qu'un sujet effectuant une action ne peut être identifié parmi un ensemble de sujets ;
- le **pseudonymat** dans lequel un sujet effectuant une action peut être identifié par un identifiant qui ne correspond pas à son identité ;
- la **non-chaînabilité** [1] dans laquelle plusieurs actions effectuées par un même sujet ne peuvent être reliées entre elles ;
- la **non-détectabilité** dans laquelle l'occurrence d'un événement ne peut être différenciée de son absence ;
- la **non-observabilité** dans laquelle un événement n'est pas détectable par les sujets non concernés et dont les sujets concernés gardent leur anonymat.

Des systèmes de gestion de l'identité peuvent aussi être mis en place pour élaborer des stratégies plus fines de fonctionnement d'un système.

Pour ce qui concerne les bases de données, des critères plus quantitatifs ont été proposés comme moyens de mesurer le niveau de respect de la vie privée, par exemple :

- le k**-anonymat** quand chaque enregistrement n'est pas distinguable d'au moins $k-1$ autres enregistrements [Swe02] ;
- la l**-diversité** quand au moins l valeurs sensibles distinctes existent pour chaque groupe k-anonyme [Mac+07] ;
- la t**-proximité** [2] quand la distribution des attributs au sein de chaque classe d'équivalence n'est pas éloignée de plus de t de la distribution de cet attribut au sein de la base de données [LLV07] ;
- l'ϵ**-différence** [3] quand la réponse à une requête ne diffère pas de plus d'un facteur ϵ qu'une personne soit présente ou non dans la base [Dwo06].

Les trois premiers critères sont cependant sensibles à des attaquants qui combinent les informations provenant de plusieurs sources. Le principal avantage de l'ϵ-différence est de fournir des garanties fortes indépendamment des connaissances auxiliaires de l'adversaire.

Ces différents critères d'analyse permettent de caractériser l'effet des outils techniques utilisés pour protéger la vie privée.

2.1.2 Technologies améliorant le respect de la vie privée

La simple promesse par les industriels ou les fournisseurs de services du respect de la vie privée des usagers n'est pas forcément suffisante. Celle-ci est parfois qualifiée de respect de la vie privée par politique [4]. Une autre piste pour respecter la vie privée est d'utiliser des outils qui empêchent les tiers de violer les politiques même s'ils le désiraient [SC09]. Beaucoup de technologies respectueuses de la vie privée [5] ont été proposées dans ce sens [Byg02 ; AGK03 ;

1. *Unlinkability* en anglais.
2. *t-closeness* en anglais.
3. *ϵ-differential privacy* en anglais.
4. *Privacy by policy* en anglais.
5. *Privacy Enhancing Technologies* (*PETs*) en anglais.

Lan+08]. Ces technologies peuvent opérer à différents niveaux dont celui du réseau comme par exemple :

— les **réseaux de mélange** [6] qui empêche la chaînabilité des messages entrants et sortants d'un routeur, généralement utilisés en série [Cha81] ;
— le **routage en oignon** [7] qui assure l'anonymat des émetteurs et destinataires d'un message à travers un routage successif de ce message chiffrés sur plusieurs couches [RSG98] ;
— les **réseaux de foules** [8] qui consistent à créer des groupes de manière à ne pas pouvoir déterminer quel est le membre du groupe à l'origine d'un message (selon une certaine probabilité) [RR98].

Ces différentes technologies, qui sont particulièrement développées pour les applications Internet [GWB97 ; Gol03 ; Gol07], reposent souvent sur des primitives cryptographiques.

2.1.3 Primitives cryptographiques

Les systèmes informatiques manipulant des données personnelles font souvent appel à des primitives cryptographiques. Celles-ci sont omniprésentes dans les services numériques dès lors qu'il s'agit de protéger des informations ou des communications.

La sécurité de ces primitives reposent sur des problèmes mathématiques qui sont supposés difficiles à résoudre sans la possession d'un secret particulier. Des clés permettent à ceux qui en ont la connaissance de retrouver l'information protégée [Sch07]. Un principe à souligner, connu sous le nom de principe de Kerckhoffs, veut que la sécurité d'un cryptosystème ne repose pas sur le secret de sa structure ou de sa configuration mais seulement sur le secret de ses clés [Ker83]. Celles-ci doivent donc être protégées pour garantir les propriétés attendues. La gestion et la protection des clés dans un système d'information est donc une condition essentielle pour assurer leur sécurité.

Plusieurs primitives cryptographiques sont souvent utilisées dans les solutions techniques de respect de la vie privée. Il s'agit notamment des primitives suivantes :

— le **schéma d'engagement** [9] qui permet à un agent de consigner une valeur sous une forme chiffrée pour la révéler plus tard. La propriété d'injectivité (théorique) de la fonction de chiffrement empêche l'agent émetteur de fournir ultérieurement une autre valeur qui a la même forme chiffrée. La valeur communiquée est ainsi fixée sans être dévoilée (immédiatement) [BCC88] ;
— le **chiffrement homomorphe** [10] qui établit un chiffrement permettant d'effectuer des calculs sur des valeurs chiffrées de telle sorte que le résultat déchiffré soit correct [RAD78]. Le chiffrement homomorphe partiel [Pai99] est utilisable en conditions réelles mais ne permet qu'un seul type d'opération. Le principal frein à l'utilisation du chiffrement homomorphe complet est aujourd'hui les temps de calculs prohibitifs qui entraînent des dégradations de performance inacceptables [Gen10] ;

6. *Mix networks* en anglais.
7. *Onion routing* en anglais.
8. *Crowds* en anglais.
9. *Commitment scheme* en anglais.
10. *Homomorphic encryption* en anglais.

— la **preuve de connaissance à divulgation nulle** [11] qui permet à une partie (le prouveur) de prouver une propriété à une autre partie (le vérifieur) sans qu'aucune autre information autre que cette propriété qui ne pouvait être déduite avant ne puisse l'être après (autrement dit, le vérifieur ne peut extraire aucune information de la preuve autre que la validité de la propriété) [GMR85] ;
— le **calcul sécurisé multipartite** [12] qui permet à plusieurs parties d'effectuer un calcul dont le résultat est connu de toutes les parties mais dont les arguments restent secrets [Yao82] ;
— le **transfert inconscient** [13] qui consiste à faire envoyer des informations d'un émetteur à un destinataire sans que l'émetteur sache quelles sont les informations qui ont été envoyées (si des informations l'ont été) ni que le destinataire n'ait accès à d'autres informations [Rab81]. Cette technique est plus protectrice de la base de données de l'émetteur que le **retrait privé d'information** [14] qui n'assure pas que le destinataire n'ait pas accès à d'autres informations [Cho+98] ;
— le **partage de secret** [15] qui consiste à distribuer des clés permettant de déchiffrer un secret entre n parties de telle sorte que m ou plus de parties, quelles qu'elles soient, puissent déchiffrer le secret et sans qu'il soit possible pour un nombre inférieur à $m-1$ de le faire [Sha79] ;
— l'**attribut certifié** [16] qui permet à une partie de prouver une propriété à une autre partie de manière anonyme [Cha85].

Cette liste ne prétend pas à l'exhaustivité : d'autres primitives existent et sont adaptées à diverses applications.

Aussi puissantes et utiles que soient ces primitives, leur bonne utilisation requiert des connaissances pointues. En supposant qu'elles soient individuellement sans faille, une des principales difficultés est leur composition. Celle-ci peut être source de nouvelles failles difficiles à détecter [Can01 ; CD09].

Le grand nombre de technologies et de primitives disponibles pour satisfaire les attentes de la société et les règlementations contraignantes rendent la conception de systèmes de plus en plus complexe. Il est naturel de se tourner vers les techniques de conception et d'analyse proposées dans le domaine du génie logiciel pour évaluer l'aide qu'elles peuvent apporter aux concepteurs dans cette tâche.

2.2 Génie logiciel

Le génie logiciel s'intéresse aux pratiques permettant de faciliter le développement de systèmes informatiques de qualité en prenant en compte toutes les contraintes qui pèsent sur cette industrie, notamment les contraintes de coût et de délai. L'approche générale repose sur l'étude du cycle de développement dont le déroulement dépend de plusieurs facteurs. Les exigences

11. *Zero-knowledge proof* en anglais.
12. *Secure multi-party computation* en anglais.
13. *Oblivious transfer* en anglais.
14. *Private information retrieval* en anglais.
15. *Secret sharing* en anglais.
16. *Anonymous credentials* en anglais.

à satisfaire jouent un rôle important dans le choix du cycle le plus adapté. Durant celui-ci, le système est modélisé de différentes façons afin de permettre différents niveaux d'analyse.

Ces différents points sont étudiés dans les sections suivantes : le cycle de développement en Section 2.2.1, le respect de la vie privée par construction en Section 2.2.2 et les niveaux de modélisation en Section 2.2.3.

2.2.1 Cycle de développement

Le cycle de développement suivi est un des éléments les plus structurants d'un projet logiciel. Il ordonne les étapes de base de l'ingénierie logicielle que sont, *a minima*, la spécification, la conception, l'implémentation, la validation et l'évolution d'un système [Som11].

Les modèles adaptés au développement d'un système dépendent de ses caractéristiques. Les principaux modèles de développement connus sont :

- la **cascade** [17] où les étapes de base sont appliquées consécutivement et de façon unitaire pour développer les fonctionnalités satisfaisant l'ensemble des exigences du système ;
- le **cycle incrémental** qui consiste à répéter le cycle de développement pour chaque fonctionnalité du système. Chaque incrément conduit au développement complet d'une fonctionnalité. Ces incréments peuvent être parallélisés si le contexte le permet. Certaines fonctionnalités peuvent ainsi être montrées au client et les éventuels problèmes détectés plus tôt dans le cycle de développement ;
- le **cycle itératif** qui consiste à répéter le cycle de développement plusieurs fois pour l'ensemble des fonctionnalités du système. Chaque itération conduit à un développement partiel de chacune des fonctionnalités du système. Le système peut ainsi être montré au client et les éventuels problèmes détectés plus tôt.

Ces trois modèles de cycle peuvent être combinés de diverses sortes. Par exemple le cycle en V [18] décompose l'étape de validation en trois sous-étapes correspondant aux trois étapes que sont l'implémentation, la conception et enfin la spécification. Plus les problèmes de validation apparaissent tardivement, plus le travail que demanderont les corrections sera important. Une autre possibilité est le cycle en spirale qui ajoute à chaque étape une analyse de risques et un choix de stratégie pour l'étape suivante visant à minimiser les risques d'échec du projet [Boe88]. Enfin, les méthodes dites agiles sont caractérisées par des cycles de développement courts reposant sur des approches à la fois incrémentales et itératives [19] [Bec+01]. Celles-ci sont particulièrement plébiscitées pour le développement d'applications *Web*. En effet, la nature changeante des exigences et donc le peu de moyens à allouer pour les définir de manière définitive en amont se prêtent naturellement à des développements « agiles ». Au contraire, la *NASA* [20] a communiqué par l'intermédiaire de Werner Gruhl sur la corrélation inverse entre les moyens alloués à une définition fine des exigences en amont du développement et le dépassement final des coûts prévus d'un programme de développement [Gru92]. Cependant, les programmes de développement de la *NASA* sont d'une nature particulière et comportent des risques, notamment financiers, sans commune mesure avec ceux des applications *Web*.

17. *Waterfall model* en anglais.
18. *V-model* en anglais.
19. *Manifesto for Agile Software Development*.
20. *National Aeronautics and Space Administration*.

Le contexte a donc une importance cruciale pour choisir le bon modèle ou la bonne combinaison et il faut analyser l'impact du respect de la vie privée sur le cycle de développement pour effectuer ce choix.

2.2.2 Respect de la vie privée par construction

Le *PbD* introduit brièvement dans l'introduction en Section 1.1.3 consiste (entre autres) à prendre en compte les exigences de vie privée dès les premières étapes du cycle de développement.

Ann Cavoukian, la commissaire à la protection des informations et de la vie privée de l'Ontario, a construit et diffusé très tôt une liste de principes à prendre en compte lors de la création d'un système collectant et traitant des données personnelles [Cav09].

Il existe d'autres visions et définitions du *PbD* [Lan01] mais toutes adoptent une approche déclarative en qualifiant les propriétés que devra respecter le système conçu plutôt que les démarches et méthodes à suivre pour y parvenir [Dia+09 ; GTD11].

Quelques groupes de travail se sont concentrés sur des aspects plus techniques relatifs au génie logiciel tels que les *Privacy Guidelines* qui proposent des bonnes pratiques pour améliorer le respect de la vie privée [MS08] ou le projet *Privacy by Design Documentation for Software Engineers* de l'OASIS [OASIS14]. Cependant, il n'existe pas encore de méthode guidant des concepteurs en leur permettant de concilier exigences fonctionnelles et exigences relatives au respect de la vie privée de manière raisonnée et systématique.

Quelques travaux dans ce sens existent cependant. Des stratégies de conception (telles que minimiser les données collectées, cacher les données des tiers, séparer les données stockées, agréger les données communiquées, informer les sujets, contrôler les traitements de données par des audits, respecter les politiques de données et démontrer leur respect) ont été proposées [Hoe14].

Dans un contexte différent, celui de la conception de systèmes d'information pour l'informatique dans les nuages, il a été proposé l'implémentation de techniques pour rendre plus facile aux développeurs la prise en compte des exigences de sécurité et de respect de la vie privée [Man+13]. Dans le même esprit, un outil d'aide à la décision basé sur des modèles de conception [21] a été proposé afin d'aider les développeurs à prendre en compte les principes de respect de la vie privée dans les étapes amont du cycle de développement [PB11].

Les divers travaux évoqués ici ciblent des niveaux de description ou des étapes différents et permettent de résoudre des problèmes différents. Un des enjeux important en la matière est de déterminer le niveau le plus adéquat pour avoir un impact important sur le développement de systèmes respectueux de la vie privée.

2.2.3 Niveaux de modélisation

Différents niveaux de modélisation existent et sont utilisés aux différentes étapes du cycle de développement. Les trois principaux sont la spécification, l'architecture et les protocoles.

21. *Design patterns* en anglais.

Le premier niveau, celui de la spécification, permet de représenter les qualités attendues d'un système sous la forme d'exigences. Il fait apparaître les différentes parties prenantes et leurs interactions fonctionnelles et non fonctionnelles avec le système. Ce niveau exprime ce qui doit être fait mais pas comment cela doit être fait. Dans une démarche de *PbD*, il est particulièrement important que les exigences soient bien spécifiées. Une erreur ou une imprécision conduira à développer un système qui, quelle que soit la qualité de la suite du développement, ne sera pas conforme à la règlementation ou aux exigences attendues.

Le niveau architectural est le premier niveau à décrire comment le système doit satisfaire les exigences. De manière générale, « [l]'architecture logicielle d'un système est l'ensemble des structures nécessaires pour raisonner sur le système, ce qui comprend les éléments logiciels, les relations entre eux et les propriétés des deux. » [22] [BCK12]. Par conséquent, l'architecture du système est une abstraction qui permet de raisonner sur les propriétés du système considéré en fonction des composants retenus. Cette étape est aussi cruciale pour la démarche du *PbD* car un mauvais choix à ce niveau, découvert à une étape tardive, peut nécessiter la remise en cause de l'ensemble de l'architecture et avoir un impact important en terme de coûts et de délais. Une contribution récente met aussi en valeur l'importance des architectures pour le respect de la vie privée dès la conception [Kun14]. Elle détaille une méthodologie de conception pour le respect de la vie privée guidée par des stratégies (telles que la minimisation, le respect, la redevabilité et l'évolutivité) et des modèles de conception (tels que le confinement de données, l'isolement ou la gestion de bases de données hippocratiques).

Les composants définis au niveau architectural peuvent alors être implémentés en faisant appel à une approche par composants [23] reposant sur des technologies et primitives présentées en Sections 2.1.2 et 2.1.3 (en prêtant une attention particulière aux difficultés de composition). Pour ce qui concerne le développement de ces composants, le niveau de modélisation adapté est celui du protocole. Ce niveau permet de détailler l'algorithme distribué ainsi que les communications effectuées par et entre les composants du système.

L'absence de faille au niveau de l'architecture n'empêche pas une faille au niveau du protocole de conduire à un non-respect des exigences du système. Une étude limitée au niveau architectural repose donc sur l'hypothèse que les protocoles sont sans défaut. Une approche mixte peut être adoptée en vérifiant certaines primitives protocolaires et en supposant que les autres sont sûres.

Quels que soient le modèle de cycle et le niveau d'analyse retenus, le respect de la vie privée pour un système est difficile à assurer et entre parfois en conflit avec d'autres exigences. Il est donc particulièrement utile de disposer de techniques de validation ou de vérification adéquats. Les méthodes formelles semblent des candidates idéales à cette fin.

2.3 Méthodes formelles

Les techniques de génie logiciel peuvent être adossées à des méthodes formelles afin d'obtenir des garanties fortes sur les résultats obtenus. Certaines méthodes sont utilisées pour vérifier que

22. « *The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.* » dans le texte.
23. *Components Off-The-Shelf (COTS)* en anglais.

le modèle d'un système respecte des propriétés spécifiques. D'autres méthodes offrent une aide supplémentaire au concepteur en lui fournissant une démarche pour construire l'implémentation en partant de la spécification. Enfin, une dernière catégorie automatise certaines étapes en dérivant automatiquement des solutions vérifiées.

Ces différentes méthodes sont détaillées dans le contexte large de la protection de données par la vérification d'un modèle en Section 2.3.1, le raffinement en Section 2.3.2 et la génération automatique en Section 2.3.3.

2.3.1 Vérification d'un modèle

La méthode de vérification d'un modèle consiste à créer un modèle et à vérifier qu'il respecte certaines propriétés.

On peut vouloir notamment vérifier la cohérence intrinsèque du modèle formel d'une spécification ou bien la satisfaction des exigences (qui doivent aussi être modélisées dans ce cas). Une fois la spécification formelle obtenue, il est alors possible de l'utiliser pour guider les phases suivantes du développement comme la vérification ou le test le produit par rapport à la spécification.

Différentes familles de méthodes existent pour modéliser et vérifier des modèles. Des langages dédiés ont été proposés pour spécifier des propriétés de respect de la vie privée [BDK04 ; Bar+06 ; BMB11 ; Jaf+11 ; LM09 ; LeM+07 ; YLA04]. Dans le même esprit, des calculs de processus comme le pi-calcul appliqué [24] ont été utilisés pour la définition de protocoles respectueux de la vie privée [AF01 ; DKR10]. Les calculs de processus sont des cadres puissants dédiés à la modélisation de systèmes concurrents. L'inconvénient de ceux-ci est donc que les protocoles exprimés dans ce formalisme sont de bas niveau et que les spécification exprimées d'une part et les propriétés à étudier d'autre part sont plus complexes que dans des langages dédiés.

La représentation et l'analyse des architectures logicielles constitue un champ de recherche actif depuis longtemps à travers les langages de description architecturale [25]. Les architectures sont souvent définies à travers des notations purement graphiques dans des cadres semi-formels ou informels. Des cadres formels outillés ont toutefois été proposés pour définir les architectures logicielles mais ils ne sont pas conçus pour exprimer des propriétés de respect de la vie privée et sont souvent trop détaillés pour être facilement reliés à de telles propriétés de haut niveau [Cle96 ; All97].

Un cadre a été proposé pour définir le sens précis des opérations disponibles dans une architecture à l'aide d'une sémantique opérationnelle associée à un système d'inférence permettant de dériver des propriétés d'architectures [LM13]. Le système d'inférence est appliqué à la preuve de propriétés de vie privée et de détectabilité de fraudes dans des systèmes de péages routiers.

Les méthodes formelles basées sur la vérification sont d'une grande aide et peuvent être utilisées de manière très souple, notamment dans la phase de validation. Il existe cependant des

24. *Applied pi-calculus* en anglais.
25. *Architecture Description Languages (ADL)* en anglais.

méthodes qui ont un impact plus fort sur le cycle de développement comme celle du raffinement par exemple.

2.3.2 Raffinement

Les méthodes par raffinement consistent à partir d'une spécification formelle et à raffiner celle-ci jusqu'au niveau de l'implémentation. Chaque raffinement est prouvé équivalent au niveau supérieur, ce qui permet ainsi d'obtenir une implémentation dont le comportement satisfait la spécification. Un des exemples les plus classiques d'une telle méthode est la méthode B et ses dérivées [Abr96]. Ces travaux ont connu un grand succès dans des applications critiques telles que le domaine ferroviaire par exemple [LSP07].

Parmi les travaux les plus importants dans cette catégorie on peut citer notamment une méthode pour développer des protocoles dont les propriétés de sécurité sont définies [SB10]. Le concepteur raffine la spécification à travers trois autres niveaux pour obtenir un modèle d'implémentation. Les preuves de correction du protocole ainsi construit sont donc développées au long du raffinement.

Une autre catégorie de méthodes contraignantes ayant un fort impact sur le cycle de développement repose sur la génération automatique de solutions satisfaisant une spécification.

2.3.3 Génération automatique

La génération automatique, ou synthèse, consiste à utiliser un outil pour générer directement une ou des solutions au problème spécifié. La principale difficulté à résoudre pour rendre cette méthode applicable est la maîtrise de la dimension de l'espace de conception.

Des travaux ont notamment été menés pour générer de manière automatique des protocoles satisfaisant des propriétés spécifiques. La méthode consiste à générer de manière systématique l'ensemble des protocoles possibles puis à filtrer ceux qui ne possèdent pas les caractéristiques recherchées. Des heuristiques existent pour couper dans l'espace de conception et éviter l'exploration systématique de branches dont aucune des solutions ne pourra s'avérer viable [SPP01].

À plus bas niveau, des travaux ont aussi été menés pour générer automatiquement des primitives du style de celles qui sont présentées en Section 2.1.3. Par exemple, des méthodes de compilation automatique de preuves à divulgation nulle de connaissance et de calculs sécurisés multipartites ont été proposées récemment [Ker12]. D'autres travaux, comme ZQL, concernent des outils permettant de compiler des protocoles comme des preuves à divulgation nulle de connaissance vérifiées et accompagnées de leurs preuves de correction [Fou+13].

2.4 Caractérisation de solutions issues de la littérature

Nous proposons dans cette section une caractérisation de plusieurs solutions proposées dans la littérature scientifique en matière de relevé intelligent de consommation électrique. Cette caractérisation prend appui sur la localisation des calculs et la manière dont les acteurs vérifient l'intégrité des données.

Nous présentons dans un premier temps les notions d'acteur, de composant et leurs rôles dans la Section 2.4.1. Deux applications différentes sont ensuite décrites : la facturation dans la Section 2.4.2 et la surveillance de réseau dans la Section 2.4.3.

2.4.1 Acteurs, composants et rôles

Un système est composé de plusieurs parties prenantes, dénommées acteurs, qui ont chacunes leurs exigences. Ces exigences peuvent entrer en conflit. Différents types d'acteurs existent comme présenté dans le Tableau 2.1 qui représente une classification inspirée de [Erk+13]. Nous avons généralisé la dénomination des rôles fonctionnels tenus par les agents pour qu'ils soient applicables à d'autres domaines que le relevé intelligent de consommation électrique.

La relation aux données exprime le lien entre les données manipulées par le système et les rôles fonctionnels : les utilisateurs (U) sont sujets des données, les fournisseurs (P) et les opérateurs (O) sont responsables de traitement [26]. Les parties manipulant des données qui ne rentrent dans aucun des rôles précédents sont dénommées sous-contractantes (SC). Les données qu'elles manipulent dépendent du service qui leur est confié. Les colonnes suivantes détaillent la finalité, la précision et le besoin d'avoir des données rattachées à un sujet identifiable pour le service mentionné. Enfin, le domaine d'agrégation exprime le domaine sur lequel porte cette opération (dans les cas où une agrégation est utilisable). Une agrégation sur les utilisateurs permet d'affaiblir l'exposition des utilisateurs en associant leurs données avec celles d'autres utilisateurs (et est donc associée à des services ne nécessitant pas de données identifiables). L'agrégation temporelle affaiblit leur exposition en diminuant la granularité temporelle des données (et donc la quantité d'informations extractibles de ces dernières). Enfin, la dernière colonne concerne le besoin d'obtention des données en temps-réel.

Ces acteurs sont instanciés par des composants qu'ils contrôlent et qui les représentent dans le système. Par conséquent, les acteurs sont vus comme ayant un rôle fonctionnel alors que les composants ont un rôle opérationnel comme détaillé dans le Tableau 2.2. Le réseau de communication et l'agrégateur mentionnés dans [Erk+13] ont été généralisés à des rôles opérationnels et apparaissent donc comme des composants.

Les modalités sont les différentes opérations qui peuvent être effectuées en fonction des rôles opérationnels. Certaines de ces modalités sont exclusives les unes des autres (une opération est soit une évaluation arithmétique soit une perturbation) alors que d'autres peuvent se cumuler (un émetteur peut envoyer une attestation à travers un canal anonyme). Le compteur apparaît comme un composant propre bien qu'il soit similaire à un calculateur équipé d'une modalité de calcul. Il tient en effet un rôle à part dans les architectures car il est directement connecté à l'environnement physique. La modalité simple pour l'émetteur et le récepteur indique une simple communication.

La différenciation des acteurs et composants selon leur rôle est générique. Différents domaines et applications peuvent s'appuyer dessus pour être caractérisés. Pour le relevé intelligent de consommation électrique, l'opérateur est le distributeur du réseau et le fournisseur est le fournisseur d'électricité qui établit la facture pour l'usager (qui correspond à l'utilisateur). La

26. Cette représentation est une simplification car les utilisateurs peuvent ne contracter directement qu'avec un fournisseur ou un opérateur qui sous-traite une partie du traitement des données à l'autre acteur.

Abbrv.	Rôle fonctionnel	Rôle juridique	Finalité	Précision	Identif.	Domaine d'agrégation	Temps-réel
U	Utilisateur	Sujet	Surveillance	Exacte	Oui		Oui
			Facturation	Exacte	Oui		Non
O	Opérateur	Responsable de traitement	Surveillance	Approx.	Non	Utilisateurs	Oui
P	Fournisseur	Responsable de traitement	Facturation	Exacte	Oui	Séries temporelles	Non
SC	Sous-contractant	Opérateur	Selon service	Selon service	Selon service	Selon service	Selon service

TABLEAU 2.1 – Classification des rôles fonctionnels tenus par les acteurs dans un système.

Abbrv.	Rôle opérationnel	Modalités
c	Calculateur	Évaluation arithmétique Agrégation Perturbation Chiffrement Anonymisation du sujet
e	Émetteur	Simple Authentification Attestation Preuve Diffusion Anonymisation de l'émetteur
r	Destinataire	Simple Vérification d'attestation Vérification de preuve
m	Compteur	Mesure
s	Hébergeur de données	

Tableau 2.2 – Classification des rôles opérationnels tenus par les composants dans un système.

facturation doit être exacte alors que les données de surveillance du réseau peuvent être approximatives aussi longtemps qu'elles sont suffisamment précises pour être techniquement suffisantes. Enfin, les agrégateurs peuvent être considérés soit comme un composant de l'utilisateur, du fournisseur ou de l'opérateur selon les cas. Dans une application de péage routier à l'usage, l'opérateur est l'entreprise assurant la gestion du traffic et le maintien de l'infrastructure tandis que le fournisseur peut être l'État (ou son organisme de perception de droits). Comme expliqué précédemment, les données relatives à la facturation doivent être liées aux utilisateurs alors que les données liées à la gestion du traffic n'en ont pas besoin.

Diverses solutions respectueuses de la vie privée ont été proposées dans ces deux domaines. Nous en proposons une caractérisation dans les sections suivantes.

2.4.2 Applications de facturation

Les architectures étudiées dans cette section se concentrent sur l'application de la facturation. Le service s'exprime sous la forme suivante :

$$\Omega = \left\{ fee_i = \sum_t \left(\mathrm{F}\left(\mathrm{S}\left(ECONS_{i,t} \right) \right) \right) \right\}$$

avec i utilisé pour désigner les consommateurs et t pour désigner les périodes temporelles. La variable $ECONS_{i,t}$ représente la consommation réelle. Les fonctions S et F représentent les fonctions de mesure et de tarification respectivement. Enfin, la fonction $\sum_t$ agrège les montants à facturer intermédiaires pour établir le montant de la facture finale fee_i.

Beaucoup de solutions reposant sur différentes architectures ont été proposées dans la littérature et sont récapitulées dans le Tableau 2.3 pour trois applications : le relevé intelligent

de consommation électrique, le péage routier à l'usage et l'assurance de véhicules à l'usage. La localisation des opérations effectuées par un composant correspondant à une fonction du service est indiquée par A(c) avec A appartenant à l'ensemble des acteurs et c à l'ensemble des composants. Cette différentiation permet de mieux séparer les rôles fonctionnels et opérationnels. Quatre manières différentes de garantir l'intégrité sont recensées : les signatures par des composants de confiance, les défis dans le cadre de schémas d'engagement, la sécurité qui repose sur des primitives cryptographiques (comme des preuves ou des calculs sécurisés) et le calcul par le composant lui-même qui s'assure ainsi de la correction du résultat.

Comme mentionné précédemment, les données agrégées de facturation doivent être identifiables et exactes. Par conséquent, peu de solutions parmi celles répertoriées utilisent des techniques d'anonymisation ou de perturbation dans leur protocole. Dans [DKR11], les perturbations introduites peuvent être déduites par l'utilisateur lorsque plusieurs factures ont été émises. Dans [Lin+12], les perturbations sont utilisées comme une forme de chiffrement dont le déchiffrement par le fournisseur est contrôlé par l'utilisateur. Enfin, la solution présentée dans [PBB09] laisse le calcul de la fonction F être effectué par le fournisseur. Cela est rendu possible tout en préservant le respect de la vie privée des utilisateurs car les données sont dé-identifiées et envoyées par un canal anonyme au fournisseur qui les rediffuse à l'ensemble des utilisateurs en retour. Les utilisateurs filtrent les données les concernant grâce à une clé d'association dont ils sont seuls détenteurs.

Les effets des contraintes spécifiques aux domaines sont visibles à travers les types de confiance choisis pour la mesure S. En effet, des compteurs certifiés qui signent les mesures sont utilisés dans le domaine du relevé intelligent de consommation électrique. Cette solution est cependant coûteuse et l'utilisation de défis dans le cadre de schémas d'engagement est préférée pour le domaine du péage routier à l'usage. En effet, des contrôles sporadiques peuvent être effectués à l'aide de portiques ou de radars mobiles alors que cela est plus compliqué pour un réseau de distribution d'électricité. Une solution technique pourrait être l'utilisation de pinces ampèremétriques. Cependant, les compteurs électriques sont souvent situés à l'intérieur des maisons et il faut alors prévoir un aménagement contractuel entre l'utilisateur et le fournisseur pour effectuer ces relevés. De plus, le fournisseur pourrait être suspecté de cibler des foyers spécifiques pour en relever le détail des consommations électriques (ce comportement déloyal ne pourrait cependant être pratiqué à grande échelle).

Enfin, la solution proposée dans [Tro+07] est atypique en ce que le compteur est chargé d'effectuer les calculs de la facturation tout en étant contrôlé par le fournisseur à l'aide de composants résistants à la falsification des données [27]. Par conséquent, l'utilisateur doit vérifier par lui-même la correction de ces calculs.

Les applications de surveillance de réseau présentent un profil différent comme nous le montrons dans la section suivante.

27. *Tamper-resistant* en anglais.

Fonction	$\sum_i$		F		S	
Choix	Intégrité	Localisation	Intégrité	Localisation	Intégrité	Localisation
Relevé intelligent de consommation électrique						
[Lin+12]	Calcul	P(c)	Calcul	P(c)	Signature	SC(m)
[LeM+07]	Signature	SC(c)	Signature	SC(c)	Signature	SC(m)
[Jes11]	Sécurité	SC(c)	Calcul	P(c)	Signature	SC(m)
[Pet10]	Signature	SC(c)	Signature	SC(c)	Signature	SC(m)
[JJK11]	Sécurité	SC(c)	Sécurité	SC(c)	Signature	SC(m)
[RD10]	Sécurité	C(c)	Sécurité ou Défis	C(c)	Signature	SC(m)
[DKR11]	Sécurité	C(c)	Sécurité	C(c)	Signature	SC(m)
[MM+10]	Sécurité	C(c)	Sécurité et Défis	C(c)	Signature	SC(m)
Péage routier à l'usage						
[PBB09]	Sécurité	C(c)	Calcul	P(c)	Défis	SC(m)
[Bal+10]	Sécurité	SC(c)	Sécurité	SC(c)	Défis	SC(m)
[JJ08]	Sécurité ou Défis	SC(c)	Défis	SC(c)	Défis	SC(m)
Assurance de véhicules à l'usage						
[Tro+07]	Calcul	P(c)	Calcul	P(c)	Calcul	P(m)
[IL07]	Signature	SC(c)	Signature	SC(c)	Signature	SC(m)

Tableau 2.3 – Localisation des calculs et vérification de l'intégrité par le fournisseur dans des applications de facturation.

2.4.3 Applications de surveillance de réseau

Nous nous intéressons dans cette section aux applications de surveillance de réseau dans les domaines de la distribution d'électricité et de la gestion de circulation routière. Dans ce cadre, le service s'exprime sous la forme :

$$\Omega = \left\{ load_t = \sum_i \left(\mathrm{S}\left(ECONS_{i,t} \right) \right) \right\}$$

où les variables et fonctions de même nom que dans la section précédente ont la même signification. La variable $load_t$ représente la charge du réseau durant la période temporelle t.

Diverses propositions de solutions respectueuses de la vie privée pour cette application ont été émises et sont récapitulées dans le Tableau 2.4.

Dans cette application où les identités relatives aux données n'ont pas besoin d'être connues et sont dé-identifiées, les opérations sont directement effectuées par l'opérateur lui-même. Le type de confiance est donc naturellement la confiance par le calcul qui est plus simple à mettre en œuvre. La dé-identification nécessite en revanche l'introduction de *PETs* supplémentaires.

Un des principaux avantages à séparer les parties et leur rôle est d'aider à rendre moins ambiguë une architecture. Par exemple, la solution de [LeM+07] est annoncée comme effectuant tous les calculs sur le compteur qui contient un module de plateforme de confiance [28]. Cependant, ce compteur n'est pas un simple compteur : il effectue des calculs, communique les données à d'autres composants et fait respecter des restrictions d'accès aux données notamment. La simple appellation de compteur peut être trompeuse pour un composant qui assume autant de rôles. Un autre avantage de cette approche est la qualification plus précise des relations de confiance : certains des travaux cités proposent des solutions reposant sur des composants sécurisés. Cependant, il n'est pas toujours indiqué clairement qui doit avoir confiance dans quels composants et pour quelles modalités.

L'équilibre d'une caractérisation doit être trouvé entre exhaustivité et lisibilité. Le bon niveau dépend de la finalité d'une telle classification. L'analyse du respect de la vie privée fait intervenir beaucoup de relations et il est difficile d'atteindre un niveau de détail permettant d'effectuer tous les raisonnements utiles pour garantir que les fournisseurs, opérateurs et utilisateurs sont assurés de la correction des données et que les exigences des consommateurs en terme de confidentialité sont respectées.

La caractérisation que nous proposons se concentre sur les calculs et ne laisse pas apparaître clairement les liens de communication entre les parties. Certaines de ces communications sont déterminées par la localisation des calculs (car les arguments doivent être communiqués pour qu'un calcul puisse être effectué). La qualification de ces communications (par canal anonyme ou par diffusion par exemple) est cependant importante [29]. Les liens de communication sont davantage mis en valeur et étudiés dans l'enquête [JKD12] pour le domaine du relevé intelligent

28. *Trusted Platform Module (TPM)* en anglais.

29. L'anonymat des canaux de communication est une des étapes de la démarche systématique (voir Section 3.5.1.3) et la diffusion peut être vue comme un canal de communication d'un composant vers tous les autres.

Fonction	$\sum_i$		S	
Choix	Intégrité	Localisation	Intégrité	Localisation
Distribution d'électricité				
[Lin+12]	Sécurité	P(c)	Signature	SC(m)
[Jes11]	Calcul	O(c)	Signature	SC(m)
[Pet10]	Calcul	O(c)	Signature	SC(m)
[JJK11]	Calcul	O(c)	Signature	SC(m)
[LLL10]	Sécurité	C_i(c)	Signature	SC(m)
[KDK11]	Sécurité	C_i(c)	Signature	SC(m)
[AC11]	Sécurité	SC(c)	Signature	SC(m)
[MM+10]	Calcul	O(c)	Signature	SC(m)
Circulation routière				
[Hoh+06]	Calcul	O(c)	Signature	SC(m)

Tableau 2.4 – Localisation des calculs et vérification de l'intégrité par le fournisseur dans des applications de surveillance de réseau.

de consommation électrique. Un des moyens de les représenter ici serait d'intégrer dans l'architecture les acteurs permettant l'implémentation de ces canaux. Ces acteurs pourraient alors représenter différentes implémentations telles que des anonymisateurs, des réseaux dédiés ou encore des réseaux pair-à-pair.

Nous voyons qu'il existe une grande quantité de solutions proposées pour un même service. Des choix ont été effectués et leur impact sur le système et ses utilisateurs n'est pas facilement observable. Ces travaux spécifiques et les solutions présentées n'expliquent pas la démarche suivie pour les obtenir.

Il n'existe pas aujourd'hui de démarche systématique et détaillée qui mettre en œuvre le *PbD* et qui :

1. servirait de support à une méthode de développement permettant d'aider un concepteur à concevoir une architecture respectant la vie privée et en particulier le principe de minimisation des données comme détaillé dans le chapitre 3 ;
2. se prête à une formalisation permettant de vérifier formellement la satisfaction des exigences comme détaillé dans le chapitre 4 ;
3. enfin, soit adossée à un outil intégré d'aide à la conception et à la vérification comme détaillé dans le chapitre 5.

L'objectif de ce travail de thèse est de répondre à ce besoin.

CHAPITRE 3

Démarche systématique de spécification, de conception et de vérification

3.1 Introduction

Le respect de la vie privée des utilisateurs dans la conception d'un système peut être perçu par les fournisseurs de solution comme une contrainte additionnelle à respecter, source de complexité, de coûts et de délais supplémentaires avant livraison. Cette contrainte deviendra toutefois une exigence légale quand le futur *GDPR* entrera en vigueur. Des solutions techniques doivent donc être proposées aux concepteurs afin de les aider dans cette tâche.

À cette fin, le respect de la vie privée des utilisateurs doit être considéré et intégré dans le cycle de développement au plus tôt, dès la définition des exigences, comme le préconise l'approche du *PbD*. Il existe quelques ensembles de principes ou lignes directrices tels que les *Fair Information Practice Principles* [RCRC73] proposés par la Federal Trade Commission, la norme ISO 29100 [ISO11], le *Privacy by Design* proposé par l'autorité de protection de l'Ontario [Cav09] ou encore la contribution sur The Future of Privacy proposée par le Working Party 29 [FoP09]. D'autres groupes de travail se sont concentrés sur des aspects plus techniques relatifs au génie logiciel tels que les *Privacy Guidelines* qui proposent des bonnes pratiques pour améliorer le respect de la vie privée [MS08] ou le projet *Privacy by Design Documentation for Software Engineers* de l'OASIS [OASIS14] pour préparer leur arrivée dans l'industrie [Not+15]. Les pratiques en terme de sécurité, qui répondent à des préoccupations plus anciennes, sont plus avancées. Divers manuels de référence [Sch96 ; And08], méthodes et modèles [BHF04 ; Sch+05 ; Haf10] et techniques d'évaluation comme les *Common Criteria* [ISO09] sont documentés dans les littératures scientifique et technique. Cependant, il n'existe pas encore de méthode guidant des concepteurs en leur permettant de concilier exigences fonctionnelles et exigences relatives au respect de la vie privée de manière ordonnée et systématique.

Nous proposons de nous concentrer sur le niveau architectural d'un système. Partant de la définition d'une architecture donnée dans le Chapitre 2, nous considérons que ses composants sont les *PETs* et que la finalité de l'architecture est leur combinaison pour satisfaire les exigences relatives au respect de la vie privée du système. Par conséquent, l'architecture nous sert d'abstraction pour raisonner sur des systèmes complexes sans que cette tâche ne soit obstruée par de trop nombreux détails. Ce niveau se prête particulièrement bien au respect de la vie privée par construction pour de multiples raisons [BCK12] :

1. L'architecture porte les choix faits au plus tôt de la conception d'un système qui s'avèrent être ceux dont le changement ultérieur est le plus difficile et le plus coûteux. De mauvais choix à cette étape peuvent grandement compromettre la satisfaction des exigences de respect de la vie privée d'un système.
2. Les architectures canalisent la créativité des concepteurs en réduisant la complexité de la conception et du système à l'étude. Grâce à leurs propriétés d'abstraction, les architectures permettent à ceux-ci de se concentrer sur les qualités essentielles d'un système, à savoir ses exigences fonctionnelles et de respect de la vie privée ainsi que la composition de technologies utilisables pour les satisfaire.
3. Une architecture documentée améliore la communication entre les parties prenantes. La documentation des choix de conception est particulièrement importante pour satisfaire les obligations émanant du Privacy Impact Assessment et du principe de redevabilité mentionnés dans le projet de *GDPR*.
4. Une architecture est créée comme un modèle transférable et réutilisable. Elle peut donc jouer un rôle clé pour améliorer la réutilisabilité de solutions respectueuses de la vie privée des utilisateurs, aboutissant à réduire les coûts et les délais de conception d'une part et à une meilleure diffusion des connaissances et des expertises chez les concepteurs d'autre part. La capacité à dépasser le cas par cas est une étape clé pour l'industrialisation du respect de la vie privée dès la conception.

L'object de ce chapitre est de proposer une méthode permettant d'exploiter une bibliothèque de composants (ou *PETs*) pour construire de manière systématique l'architecture d'un système respectant une propriété essentielle en matière de protection de la vie privée, à savoir la minimisation des données. Une vue d'ensemble du cycle de développement est donnée en Section 3.2. Les hypothèses de travail retenues sont présentées en Section 3.3 avant la méthode elle-même dans les Sections 3.4 et 3.5.

3.2 Cycle de développement

La méthode proposée ici répond à deux objectifs principaux :

— aider un concepteur de système dans ses choix pour construire une architecture en minimisant les données personnelles divulguées ;
— motiver les choix effectués par le concepteur en vue de la documentation du système.

Un défi majeur réside dans la combinatoire à laquelle fait face le concepteur. Pour des systèmes réels, l'espace de conception est très vaste et la présence d'une méthode pour guider le concep-

teur est essentielle. Peu de solutions existent aujourd'hui pour aider le concepteur dans cette tâche. Différentes voies sont envisageables :

L'étude de solutions ad-hoc. Proposées dans la littérature scientifique et technique, de telles solutions existent en grand nombre pour divers domaines d'applications. La difficulté pour un concepteur non spécialiste est de trouver une solution dont les hypothèses se rapprochent de ses contraintes ou de s'inspirer de différentes solutions pour en construire une adaptée à ses besoins. À titre d'exemple, une enquête [JKD12] étudie 23 solutions différentes pour concevoir un système de capteurs électriques intelligents et les compare selon des critères fonctionnels et techniques. Cependant, ce type de document n'existe pas pour tous les domaines d'application et ne peut remplacer une méthode.

La composition de schémas de conception. Des patrons spécifiques au respect de la vie privée ont été proposés dans la littérature [Sch+05 ; Haf10 ; PB11]. Cependant, ceux-ci portent uniquement sur certaines catégories d'application comme des interfaces *Web* ou des technologies de sécurisation des données par exemple.

La synthèse d'architectures. Cette approche a pour inconvénient l'explosion combinatoire. Les possibilités d'architectures sont nombreuses et le processus de génération doit être borné. Cette approche a été explorée au niveau protocolaire par [Son99] qui propose un générateur de protocoles associé à un filtre qui ne retient que les propositions conformes. Nous n'avons pas connaissance d'une telle approche au niveau architectural.

La programmation par contraintes. Des approches systématiques reposant sur des techniques comme la satisfaction booléenne ou la résolution de contraintes nécessitent une expertise dans des outils techniques. Ceux-ci sont généralement appliqués à des domaines bien définis (optimisation d'emplois du temps, planification logistique, ...) et nous ne connaissons pas de solutions ni de prototypes d'outils pour le *PbD* reposant sur la programmation par contraintes [1].

La démarche systématique proposée dans ce chapitre s'étend de la spécification fonctionnelle du système à la conception de son architecture comme le montre la Figure 3.1. Les exigences du système peuvent être établies par le concepteur et le donneur d'ordres et la conception de l'architecture est effectuée par le concepteur.

Bien que le but de cette méthode soit de permettre la convergence la plus rapide possible vers une ou des solutions, il peut arriver de devoir revenir en arrière dans le cycle. Ces itérations se produisent lorsqu'aucune solution satisfaisante n'apparaît et qu'il est souhaitable de revenir sur des choix passés en assouplissant une exigence, en introduisant de nouvelles données, en enrichissant le modèle du service ou en ajoutant des entités. Par exemple, il peut être nécessaire d'introduire un tiers de confiance dans une spécification. Les itérations prennent fin dès l'obtention d'une architecture satisfaisant la spécification. Les choix à remettre en cause doivent être sélectionnés par leur capacité à ouvrir la voie vers de nouvelles solutions tout en minimisant l'impact sur la solution en cours de développement.

1. Nous avons exploré cette piste pour optimiser la localisation des calculs afin de minimiser les données divulguées [ALM12] mais avons préféré l'approche décrite dans ce chapitre

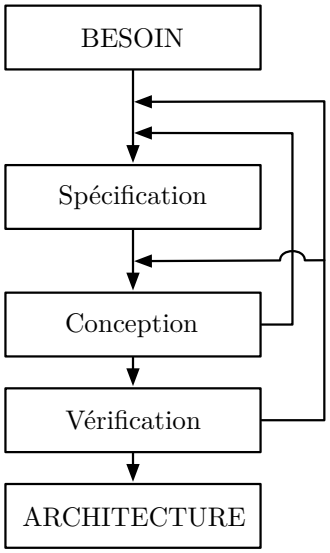

Figure 3.1 – Cycle de développement, détaillé dans les Figures 3.2, 3.5 et 3.11.

Davantage de détails sur la spécification et la conception de l'architecture sont donnés respectivement en Sections 3.4 et 3.5. Un exemple est présenté au fil de l'eau afin de montrer plus clairement comment s'appliquent les différentes étapes. Nous présentons au préalable les hypothèses de travail retenues pour ce travail dans la section suivante.

3.3 Hypothèses

Selon le principe de minimisation des données [2], nous considérons que tous les acteurs impliqués, mis à part le sujet lui-même, sont des attaquants potentiels. Nous nous intéressons aux données auxquelles ces acteurs, représentés par des composants, peuvent avoir accès à travers les communications et calculs permis par l'architecture construite en retenant les règles suivantes :

- pour les communications :
  - une donnée communiquée est accessible à l'expéditeur et au destinataire ;
  - une donnée ne peut être communiquée que par un canal de communication ;
- pour les calculs :
  - un (ou un groupe de) composant(s) qui effectue un calcul a accès aux arguments nécessaires à ce calcul ;
  - un composant participant à un calcul a accès au résultat de ce calcul [3] ;
  - une donnée ne peut être calculée que par un (ou un groupe de) composant(s) muni(s) de la capacité de le faire.

2. Dans le sens où tout acteur doit n'avoir accès qu'aux données strictement nécessaires à la poursuite de la finalité.

3. Nous nous plaçons donc dans le cas où les primitives de calcul sécurisé multipartite utilisées garantissent la propriété d'équité (*fairness* en anglais).

Nous ne considérons donc pas ici la sécurisation des canaux qui est supposée effectuée par d'autres moyens. L'authentification des composants est également assurée par des moyens qui sortent du cadre de cette étude.

Nous adoptons une hypothèse retrouvée classiquement dans le modèle d'attaquant dit de Dolev-Yao [DY81] : les primitives cryptographiques et les technologies améliorant le respect de la vie privée utilisées dans l'architecture sont supposées parfaites. Une fonction de hachage ne peut être inversée et une fausse preuve ne peut convaincre un vérifieur par exemple. Tout accès interdit à une variable est donc le résultat d'un problème de composition des briques de base, et non des briques elles-mêmes. Nous considérons donc des acteurs honnêtes mais curieux : les protocoles sous-jacents sont réputés inviolables et les acteurs ne peuvent les violer. En revanche, ils peuvent chercher à exploiter au maximum les informations qu'ils reçoivent en effectuant tous les calculs et communications dont ils sont capables. Les hypothèses ci-dessus reflètent le fait que nous avons fait le choix ici de nous focaliser sur les questions spécifiques à la protection de la vie privée en considérant que les propriétés de sécurité traditionnelles (confidentialité des contenus et authenticité des acteurs lors des communications par exemple) sont assurées par des moyens qui sortent du cadre de cette étude.

Nous détaillons maintenant les étapes de spécification et de conception d'architecture dans les sections suivantes.

3.4 Spécification

La spécification, première étape à mener à bien, définit l'objectif à atteindre pour le donneur d'ordres et le concepteur du système. Elle se décompose en trois étapes comme illustrées sur la Figure 3.2. Ces étapes sont le recensement des acteurs en Section 3.4.1, la modélisation du service en Section 3.4.2 et la définition des exigences en Section 3.4.3.

Nous détaillons chacune de ces étapes dans ce qui suit. Celles-ci se concentrent sur la spécification fonctionnelle. Des éléments de la spécification non fonctionnelle interviendront toutefois au cours de la conception pour effectuer des choix.

3.4.1 Recensement des acteurs

Les acteurs du système sont toutes les parties prenantes, notamment les sujets des données, les responsables de traitement et, enfin, les opérateurs de traitement. Nous nous appuyons sur les trois types d'acteur (sujet, responsable de traitement et opérateur) définis explicitement dans le projet de *GDPR* comme présenté en Section 1.1.3.

Le principe de minimisation des données impose de minimiser non seulement les données collectées mais aussi les acteurs qui peuvent y avoir accès.

Les acteurs qui n'entrent pas dans les catégories précédentes sont considérés comme des tiers selon nos hypothèses. Les protocoles et primitives cryptographiques étant supposés parfaits, les tiers ne peuvent ni interagir avec les acteurs légitimes ni intercepter des données. En pratique, toute mise en œuvre d'une architecture construite selon la méthode proposée ici devra donc intégrer des moyens d'assurer ces hypothèses (par des mesures de sécurité notamment).

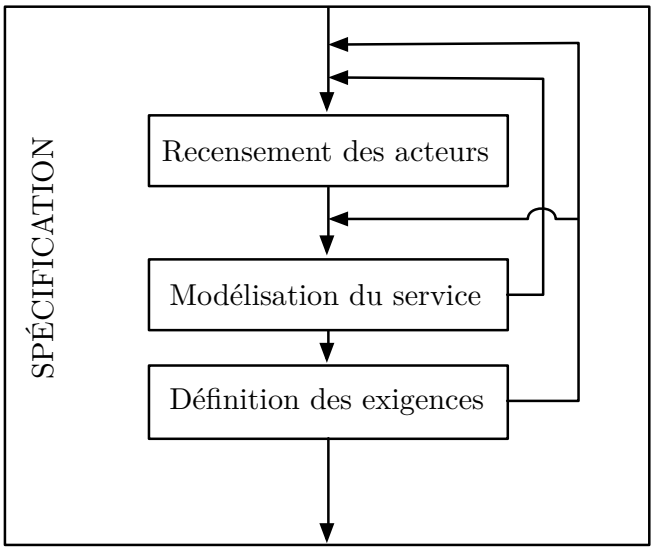

Figure 3.2 – Détail du cycle de spécification de la Figure 3.1.

Exemple. Nous prenons pour exemple la facturation de consommation d'électricité reposant sur des compteurs intelligents. Ce domaine d'application est d'actualité et la CNIL [4] a émis trois scénarios autour du relevé intelligent de consommation électrique [CF14]. En effet, ces compteurs se caractérisent par des relevés de consommation très fréquents, ce qui peut s'avérer intrusif pour les usagers (voir [Liu+12]). Deux acteurs sont nécessaires pour modéliser le service et illustrent deux types principaux d'entités décrits précédemment. Les usagers sont naturellement les sujets des données personnelles relatives à leur propre consommation électrique et le fournisseur d'électricité agit comme responsable de traitement (pour calculer le prix à facturer à l'usager à la fin de la période de facturation).

Nous supposons que tous les usagers ont un comportement similaire. Ils sont donc indistinctement désignés par $U \in \mathcal{DS}$ avec U un identifiant unique. Le fournisseur est désigné par $P \in \mathcal{DC}$.

Nous supposons dans cet exemple que le concepteur n'a pas à ce stade d'idée préconçue sur l'architecture du système et que les deux seuls types d'acteurs envisagés sont les usagers et les fournisseurs.

3.4.2 Modélisation du service

Une fois les acteurs recensés, le service est modélisé sous la forme d'un ensemble d'équations. Les variables utilisées représentent les données d'entrées (réputées correctes par définition), des variables intermédiaires, ou des résultats. Les fonctions représentent des opérations comme des mesures ou des calculs. Nous considérons par défaut que les fonctions sont facilement inversibles (c'est-à-dire qu'il est facile de retrouver les arguments à partir du résultat), ce qui

4. Acronyme de la Commission Nationale de l'Informatique et des Libertés.

correspond à l'hypothèse la plus défavorable pour ce qui est de la minimisation. Le concepteur devra donc indiquer expressément qu'une fonction n'est pas inversible.

Puisqu'elles représentent un service à accomplir, ces équations doivent représenter une chaîne d'opérations sur des variables. Le service doit donc aussi pouvoir être mis sous la forme d'un graphe orienté acyclique dont les feuilles représentent les variables et les nœuds des fonctions appliquées à leurs fils. Nous munissons ce graphe de nœuds intermédiaires permettant de nommer le résultat des opérations afin de faciliter leur désignation ultérieure. Les variables d'entrée sont représentées par des nœuds dont le contour est doublé.

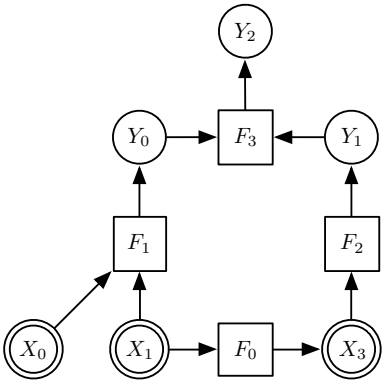

FIGURE 3.3 – Exemple de représentation graphique d'un modèle de service.

Le graphe représenté Figure 3.3 décrit un service Ω qui peut aussi s'exprimer sous la forme équationnelle :

$$\Omega = \left\{ \begin{array}{c} Y_2 = F_3\left(Y_0, Y_1\right) \\ Y_1 = F_2\left(X_3\right) \\ Y_0 = F_1\left(X_0, X_1\right) \\ X_3 = F_0\left(X_1\right) \end{array} \right\}$$

où X_0, X_1 et X_3 sont les données d'entrée du système. Dans ce cas, l'équation $X_3 = F_0\left(X_1\right)$ n'est pas une opération produisant une nouvelle variable mais une simple relation de cohérence de l'environnement.

Nous supposons que la modélisation du service est publique : les relations entre les variables et les fonctions qui la composent sont connues de tous les acteurs. En d'autres termes, chacun des acteurs connaît la spécification fonctionnelle du système. En revanche, les valeurs des variables ne le sont pas forcément et font l'objet de l'étude des propriétés de confidentialité.

Exemple. Dans notre exemple, le service se modélise comme un montant à facturer, qui est la somme des tarifs correspondant aux consommations à chaque instant.

Afin d'exprimer les unités de temps, nous introduisons l'indice t borné par le nombre de périodes temporelles retenues dans la période de facturation. Nous pouvons alors modéliser le

service Ω_0 comme suit :

$$\Omega_0 = \left\{ \begin{array}{c} fee = \sum\limits_t (price_t) \\ price_t = \mathrm{F}\,(cons_t) \end{array} \right\}$$

avec F une politique tarifaire permettant d'obtenir le tarif $price_t$ pour une consommation particulière $cons_t$ de l'usager U à l'instant t. La version graphique de ce service est représentée en Figure 3.4.

Tous les acteurs, à savoir les usagers i et le fournisseur d'électricité P connaissent le service et peuvent s'en servir pour calculer ou raisonner.

Figure 3.4 – Modèle du service Ω_0.

Dans cet exemple, on peut noter que $cons_t$ est une variable de l'environnement. Nous supposons que $\sum$ est une fonction agrégative et que F est une fonction générique supposée facilement inversible par défaut.

3.4.3 Définition des exigences

Une fois le modèle exprimé, il devient possible d'établir les exigences des acteurs vis-à-vis des données à manipuler. Celles-ci se divisent en deux catégories distinctes :

La confidentialité. Ce type d'exigence établit les liens d'accès entre entités et données. La loi Informatique et Libertés [Loi78], la directive européenne relative à la protection des données personnelles [Dir95], et le projet de *GDPR* [GDPR14] donnent aux sujets le droit d'accéder à toutes les données à caractère personnel qui les concernent. Cette contrainte doit être prise en compte dès la définition des exigences. Les parties tierces ne doivent avoir accès qu'aux données nécessaires pour remplir la finalité déclarée. Par défaut, conformément au principe de minimisation des données, on considère que l'accès à une variable est interdit quand il n'a pas été explicitement requis dans les exigences. Il n'est donc pas utile de spécifier explicitement les interdictions d'accès. En revanche, il est nécessaire de recenser les accès autorisés puisqu'ils dérogent au principe de minimisation.

L'intégrité. Ce type d'exigence définit le type d'assurance qu'une entité doit obtenir quant à la correction d'une variable (par rapport à la spécification). Cette exigence se décline en deux niveaux : la connaissance (certaine) et la croyance. La croyance correspond aux cas où l'intégrité n'a pas été complètement vérifiée mais donne toutefois une assurance élevée d'intégrité.

Ces deux dimensions, confidentialité et intégrité, sont traitées de manière orthogonale : aucune des deux n'implique l'autre.

Exemple. Dans notre exemple, à la fois l'usager U et le fournisseur d'électricité P doivent avoir accès aux montants fee_i concernant l'usager. De plus, l'usager U doit avoir accès à ses données personnelles $cons_t$ et $price_t$. En revanche, le fournisseur d'électricité P ne doit pouvoir accéder ni aux variables $cons_t$ ni aux variables $price_t$ car la fonction de tarification F, connue de tous, est jugée facilement inversible.

Pour ce qui est de l'intégrité, l'usager comme le fournisseur doivent avoir confiance (ou « connaissance ») dans la véracité des relations $fee = \sum_t (price_t)$ et $price_t = \mathrm{F}(cons_t)$ (sans que cela n'implique que le fournisseur ait accès aux variables $cons_t$ et $price_t$).

Pour récapituler, les exigences de l'exemple sont résumées dans les Tableaux 3.1 pour la confidentialité et 3.2 pour l'intégrité.

Variables	Accès autorisé	Accès interdit
$cons_t$	U	P
$price_t$	U	P
fee	U, P	

Tableau 3.1 – Spécification des exigences de confidentialité pour Ω_0.

Relations	Connaissance
$fee = \sum_t (price_t)$	U, P
$price_t = \mathrm{F}(cons_t)$	U, P

Tableau 3.2 – Spécifications des exigences d'intégrité pour Ω_0.

La définition des exigences clôt cette section et permet au concepteur de passer à la phase de conception proprement dite de l'architecture, objet de la section suivante.

3.5 Conception architecturale

L'étape suivant la spécification fonctionnelle est la conception d'une architecture qui la satisfait. Les étapes de conception sont présentées dans la Figure 3.5.

La méthode proposée ici, basée sur une succession d'interactions, aide le concepteur à trouver une voie parmi la variété d'options possibles. Chaque étape consiste en une question dont la réponse (apportée par le concepteur) permet de réduire l'espace de conception possible pour converger vers une solution satisfaisante. Les ingrédients affectant l'efficacité de ce processus sont les critères retenus et l'ordre dans lequel les questions doivent être posées. Sur la base de notre expérience des facteurs clés dans la conception de solutions architecturales respectueuses de la vie privée, nous proposons la démarche présentée dans les sections suivantes.

Les critères retenus pour guider le concepteur se divisent en deux groupes. Dans un premier temps, des choix dont l'effet agit sur la structure globale de l'architecture sont passés en revue en Section 3.5.1. Dans un second temps, une suite d'étapes est appliquée à chaque équation du service comme décrit en Section 3.5.2 pour rendre l'architecture plus opérationnelle avant de la finaliser comme détaillé en Section 3.5.3.

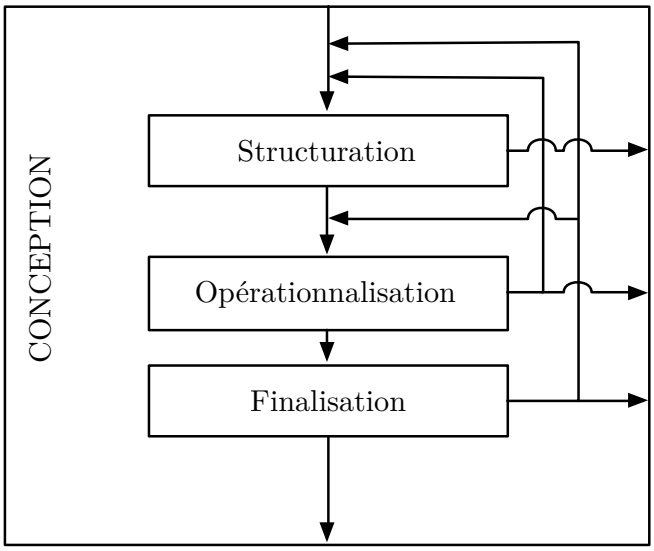

Figure 3.5 – Détail du cycle de conception de la Figure 3.1, détaillé dans les Figures 3.6, 3.7 et 3.9.

3.5.1 Structuration

La phase de structuration comprend les étapes ayant un impact majeur sur l'architecture. Celles-ci permettent d'éliminer rapidement un grand nombre d'options incompatibles avec les exigences à satisfaire. Ces étapes sont réparties en trois groupes : la détermination des composants principaux (Section 3.5.1.1), la prise en compte des choix imposés (Section 3.5.1.2) et la question de la nécessité d'anonymat (Section 3.5.1.3) comme représentés dans la Figure 3.6.

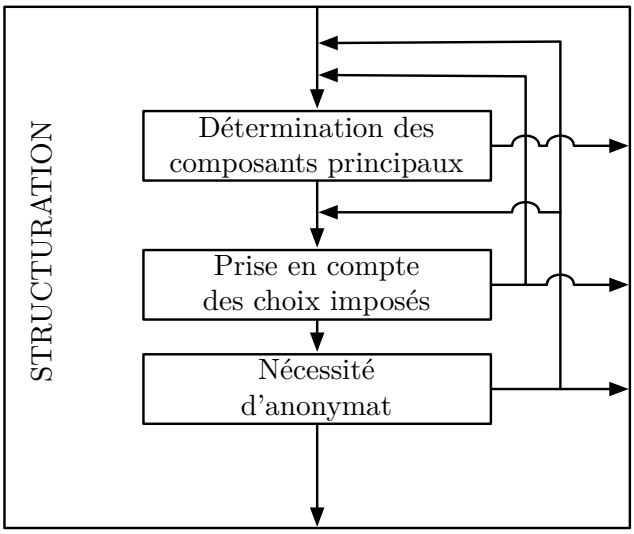

Figure 3.6 – Détail du cycle de structuration de la Figure 3.5.

3.5.1.1 Composants participants

Les composants sont directement reliés aux acteurs qui les contrôlent, au sens où ces derniers déterminent leur comportement. En revanche, les acteurs ne peuvent pas forcément communiquer avec ceux-ci ni avoir accès aux données qu'ils manipulent.

Tous les acteurs doivent contrôler au moins un composant au sein du système. Dans le cas contraire, un acteur n'aurait aucune possibilité d'agir au sein du système : il ne serait donc plus une partie prenante et deviendrait un tiers.

Toujours dans une logique de minimisation des données, il est préférable de partir d'une quantité réduite de composants, le minimum étant un composant par acteur, pour en ajouter ensuite en cas de nécessité. Lors des itérations suivantes, l'ajout de nouveaux composants, voire de nouveaux acteurs, peut servir à assouplir une spécification pour laquelle aucune architecture ne peut être conçue.

Exemple. Dans notre exemple, nous considérons que chaque acteur contrôle un composant dans l'architecture. Le système comprend donc un composant C_{M} pour le compteur, un composant C_{U} pour l'usager et un composant C_{P} pour le fournisseur d'électricité.

3.5.1.2 Choix imposés

Certains choix peuvent être imposés soit par le contexte de déploiement de l'architecture, soit par le donneur d'ordres ou encore par le concepteur lui-même quand il a déjà réalisé certains choix de conception *a priori*.

Par exemple, l'emplacement des variables d'environnement est déterminé par les capteurs qui les mesurent. L'absence de canal de communication entre certains composants peut aussi être une contrainte forte, soit parce que ces canaux ne pourraient pas être mis en œuvre, soit parce qu'ils est nécessaire de garantir l'isolement des composants. Les capacités de calcul sont aussi une source de choix imposés : des compteurs ou des unités mobiles ne sont pas forcément capables d'effectuer des calculs complexes et ces limites doivent être prises en compte.

Exemple. Dans notre exemple, les $cons_{i,t}$ sont des grandeurs qui doivent être mesurées. Une première opération à effectuer est la mesure à l'aide d'un instrument métrologique comme un compteur C_{M}. Ceux-ci sont certifiés par une tierce partie. Il est donc nécessaire de revenir en arrière dans le cycle de conception pour ajouter un tel acteur $\mathrm{M} \in \mathcal{DP}$. Ces composants ne peuvent qu'avoir accès aux données qu'ils mesurent, soit les $cons_{i,t}$. La tarification F et le mode de calcul de la facture finale $\sum_t$ étant publics, les compteurs ont donc aussi accès aux tarifs individuels $price_{i,t}$ et au montant de facturation final fee_i.

3.5.1.3 Anonymat

Une autre exigence importante est le besoin d'anonymat au sein du système. Celle-ci dépend fortement du service rendu : certains, comme la facturation, nécessitent de pouvoir relier des données à des acteurs alors que d'autres, comme la surveillance d'un réseau, peuvent se contenter de données anonymisées. Pour assurer l'anonymat, il est nécessaire d'utiliser des

canaux anonymes pour transmettre l'information entre composants au sein de l'architecture. Un canal anonyme empêche le destinataire de connaître l'expéditeur d'une donnée. Les canaux de communication sont supposés non anonymes par défaut, cette approche évite qu'un oubli du concepteur conduise à un résultat faussement satisfaisant.

Exemple. Le service de facturation de notre exemple est par définition nominatif. Il ne comporte donc pas d'exigence d'anonymat.

3.5.2 Opérationnalisation

À cette étape, les critères les plus structurants pour une architecture ont déjà été choisis. Dès lors, l'objectif est ici de permettre au concepteur de converger vers une proposition d'architecture. Les équations du service définies dans la Section 3.4.2 sont traitées tour à tour en partant des équations contenant des variables d'entrée et en remontant vers celles contenant les nouvelles variables calculées. Ce procédé s'avère le plus efficace car la localisation des variables d'entrée est généralement imposée par le contexte et celles-ci constituent souvent des données personnelles.

À ce stade, chaque équation du service (dont toutes les variables ne sont pas des variables d'entrée) définie dans la spécification est instanciée par une opération dans l'architecture. Ces opérations peuvent représenter des réalités distinctes comme des mesures, des calculs ou encore des affaiblissements de données (comme une perturbation ou un échantillonnage par exemple). Nous montrons dans les sections suivantes comment associer les opérations aux composants (Section 6.3.2.1), affaiblir les données (Section 3.5.2.2) et choisir un type de confiance approprié (Section 3.5.2.3).

La Figure 3.7 présente ces étapes dans le cycle de conception.

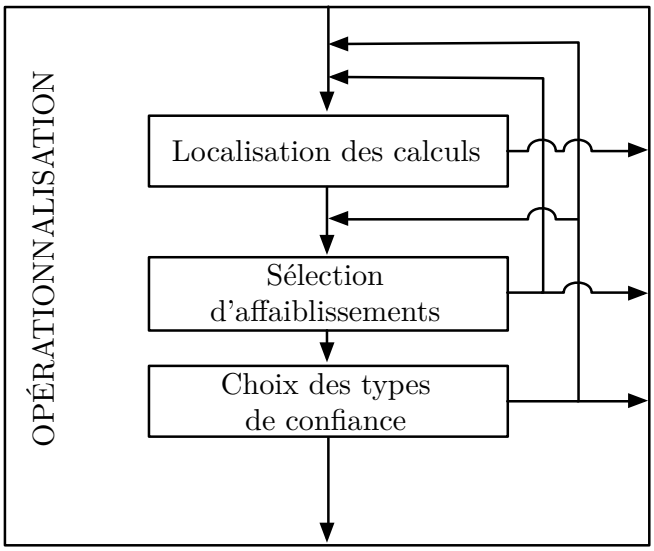

Figure 3.7 – Détail du cycle d'opérationnalisation de la Figure 3.5.

3.5.2.1 Localisation des calculs

Les calculs à effectuer apparaissent explicitement dans le modèle du service. Les variables d'entrée doivent être mesurées de façon à transformer les quantités physiques en quantités numériques. Ces mesures sont effectuées par des composants métrologiques généralement certifiés par une partie tierce. Cette différence n'a pas d'impact sur leur représentation et leur traitement dans notre démarche.

Celle-ci a un impact sur la divulgation des données personnelles et doit être choisie afin de la minimiser. Pour ce faire, deux types de stratégies peuvent être appliquées :

Maximiser la décentralisation du calcul. Cette stratégie consiste à effectuer les calculs au plus près de la source de leurs arguments, ce qui permet de limiter leur divulgation. En général, les arguments ne se trouvent pas forcément tous sur le même composant, certaines communications de ces arguments entre composants seront nécessaires.

Maximiser le nombre de parties du calcul. Cette stratégie implique d'utiliser un calcul sécurisé multipartite : celui-ci permet à plusieurs composants d'effectuer un calcul dont le résultat est connu de tous mais dont les paramètres ne sont pas partagés (par opposition à un calcul simple où un seul composant effectue le calcul).

Le choix du type de stratégie s'effectue selon trois critères principaux :

La confidentialité. Le calcul simple, qui est un calcul dans lequel un seul composant intervient, nécessite la communication de tous les arguments au composant chargé d'effectuer le calcul. Le calcul sécurisé multipartite, en revanche, permet à chaque composant participant de garder secret ses arguments (mais l'ensemble des arguments doivent être connus par ce groupe de composants). Dans tous les cas, les composants participants au calcul ont connaissance du résultat.

L'intégrité. L'acteur qui contrôle un composant a confiance dans les calculs qu'il effectue. En revanche, si un calcul est effectué sur un composant non contrôlé par un acteur, alors ce dernier devra déployer d'autres stratégies pour se convaincre de l'intégrité du résultat. Tous les participants à un calcul sécurisé multipartite savent que le résultat calculé est correct.

Le contexte. Le contexte impose des contraintes qui peuvent limiter les choix possibles. Pour qu'un calcul distribué puisse être effectué, le composant devra avoir la capacité d'effectuer ce calcul et se faire communiquer les données nécessaires. Le frein à l'utilisation d'un calcul sécurisé multipartite est surtout le coût élevé en calcul et en communications. Le concepteur doit aussi tenir compte de l'acteur associé au composant et de son rôle dans le système pour choisir la localisation d'un calcul. Par exemple, un composant associé à un fournisseur de prestations de calculs peut être privilégié pour effectuer des calculs complexes contrairement à un composant déjà présent dans l'architecture mais disposant de faibles capacités comme un compteur.

Ces critères entrent généralement en tension les uns avec les autres. Par exemple, un calcul simple effectué chez un composant contrôlé par un fournisseur est la manière la plus simple

pour ce fournisseur d'être certain de l'intégrité du résultat. En revanche, c'est aussi la solution la plus intrusive pour l'utilisateur. Au contraire, un calcul simple effectué chez un composant contrôlé par un utilisateur est un choix très protecteur de sa vie privée mais *a priori* peu certain pour le fournisseur. Les méthodes permettant de convaincre un acteur de l'intégrité d'un résultat effectué sur un composant qu'il ne contrôle pas sont présentées dans la Section 3.5.2.3 sur les types de confiance.

Exemple. Nous rajoutons parmi les équations du modèle l'équation modélisant la mesure par les compteurs. Nous obtenons donc un nouveau modèle de service Ω_1 tel que :

$$\Omega_1 = \left\{ \begin{array}{c} fee = \sum_t (price_t) \\ price_t = \mathrm{F}\,(cons_t) \\ cons_t = \mathrm{S}\,(ECONS_t) \end{array} \right\}$$

avec $ECONS_t$ la consommation réelle issue de l'environnement. Cette dernière devient la variable d'entrée du service comme représentée en Figure 3.8.

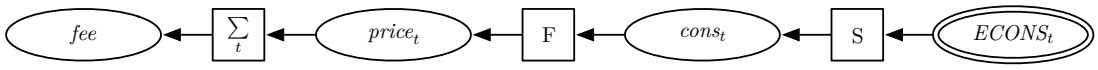

Figure 3.8 – Modèle du service Ω_1.

$ECONS_t$ est rajoutée au Tableau 3.1 pour donner le Tableau 3.3.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	U	P
$cons_t$	U	P
$price_t$	U	P
fee	U, P	

Tableau 3.3 – Spécification des exigences de confidentialité pour Ω_1.

L'équation faisant intervenir $ECONS_t$ est de la même façon rajoutée au Tableau 3.2 pour donner le Tableau 3.4. Le calcul correspondant à cette première fonction est localisé au sein des composants C_M qui représentent les compteurs : ce choix est imposé par le contexte.

Relations	Connaissance
$fee = \sum_t (price_t)$	U, P
$price_t = \mathrm{F}\,(cons_t)$	U, P
$cons_t = \mathrm{S}\,(ECONS_t)$	U, P

Tableau 3.4 – Spécifications des exigences d'intégrité pour Ω_1.

La deuxième fonction rencontrée dans la spécification de notre exemple est F. Comme les données de consommation sont déjà connues du compteur C_M, l'option privilégiée consiste à

localiser le calcul F au même endroit. Cependant, cette hypothèse n'est pas forcément réaliste car les compteurs ont en général des capacités de calcul limitées et ne servent qu'une seule fonction qui est celle de la mesure (S ici). La deuxième option pour effectuer ce calcul est alors le composant C_U contrôlé par l'usager U (en pratique, il peut s'agir de son *PC* ou d'un équipement dédié). C'est l'option du calcul par le composant de l'usager C_U qui est retenue.

La dernière fonction apparaissant dans la spécification est la somme $\sum_t$ des consommations individuelles $price_t$. Le meilleur endroit pour effectuer le calcul correspondant est également le composant C_U puisqu'il a déjà accès aux arguments nécessaires à cette fin.

Les choix de localisation pour les trois fonctions S, F et $\sum_t$ sont rappelés dans le Tableau 3.5.

Calcul	Localisation
$fee = \sum_t (price_t)$	C_U
$price_t = F(cons_t)$	C_U
$cons_t = S(ECONS_t)$	C_M

TABLEAU 3.5 – Localisation des calculs pour Ω_1.

3.5.2.2 Sélection d'affaiblissements

L'affaiblissement d'une donnée consiste à réduire sa précision selon une méthode prédéfinie en minimisant l'incidence sur la qualité du service.

Les affaiblissements n'apparaissent pas forcément dans la spécification initiale du service modélisé. Ils peuvent cependant se déduire des niveaux de qualité exigés pour les services considérés. Ces niveaux de qualité doivent être définis à l'étape de spécification ou alors lors d'itérations ultérieures. Certains services (comme la facturation) nécessitent des résultats exacts alors que d'autres peuvent se satisfaire d'approximations (comme la surveillance en temps réel d'un réseau). Nous considérons ici quatre types d'affaiblissements différents : l'agrégation, l'échantillonnage, la généralisation et la perturbation. Le choix entre ces différentes options s'effectue selon les critères suivants : le caractère déterministe du résultat, la conservation de la précision des arguments, la conservation du cardinal des arguments et l'inversibilité de l'affaiblissement. La spécification non fonctionnelle permet au concepteur de choisir les options les plus adaptées en fonction du contexte. Les correspondances entre ces critères et les types d'affaiblissement sont résumées dans le tableau 3.6. Nous les présentons de manière plus détaillée dans les paragraphes suivants.

Agrégation. L'agrégation de données est une opération visant à ne retenir qu'une seule donnée pour représenter un ensemble.

Pour ce faire on peut avoir recours à des opérations comme la moyenne, la médiane, le minimum, le maximum ou encore le cardinal des éléments d'un ensemble. Par ailleurs, la connaissance du résultat de l'agrégation peut parfois permettre de déduire des informations sur l'ensemble de départ, notamment quand des informations auxiliaires sont disponibles.

Échantillonnage. L'échantillonnage consiste à ne garder qu'une partie des données d'un ensemble.

Cette opération n'est pas nécessairement déterministe. Elle produit des valeurs moins nombreuses mais exactes. Par ailleurs, elle n'est pas inversible car il n'est pas possible de retrouver l'ensemble des valeurs de l'ensemble de départ.

Généralisation. La généralisation ne diminue pas le nombre de données mais modifie ces dernières. Dans le cas d'opérations sur des nombres, chaque valeur sera remplacée par la valeur la plus proche, l'arrondi supérieur ou encore l'arrondi inférieur.

Cette opération est déterministe et conserve le nombre de données de l'ensemble de départ mais elle ne conserve pas l'exactitude de ces données. Enfin, il est possible, pour chaque valeur de l'ensemble d'arrivée, de retrouver l'intervalle dans lequel se situe la valeur correspondante de l'ensemble de départ.

Perturbation. La perturbation consiste à modifier une valeur de manière aléatoire en suivant une loi probabiliste définie et paramétrée *a priori*.

Cette opération n'est donc pas déterministe et ne conserve pas l'exactitude des données. Par ailleurs plusieurs applications produisent des résultats différents en raison de l'aléa. En revanche, l'opération ne réduit pas le nombre de données et permet de reconstruire une distribution probabiliste pour les données de départ.

Affaiblissement	**Déterminisme**	**Conservation de l'exactitude**	**Conservation du cardinal**	**Inversibilité**
Agrégation	oui	oui	non	intervalles bornes
Échantillonnage	non	oui	non	extrapolations
Généralisation	oui	non	oui	intervalles
Perturbation	non	non	oui	distributions

Tableau 3.6 – Critères pour le choix du type d'affaiblissement.

Si le concepteur décide d'introduire une opération d'affaiblissement, celle-ci doit être ajoutée au service. L'affaiblissement ajouté à l'architecture s'interprète alors comme une opération d'un type particulier.

Exemple. L'application retenue ici est une application de facturation. La règlementation exige que cette donnée soit précisément établie car la facture doit refléter exactement la consommation de l'usager. Nous supposons donc qu'aucune technique d'affaiblissement n'est utilisée.

3.5.2.3 Choix des types de confiance

Comme décrit précédemment, la spécification de l'architecture se présente sous la forme d'une succession d'applications de fonctions à des arguments, ceux-ci étant eux-mêmes soit des

résultats d'applications de fonctions à d'autres arguments, soit des variables d'environnement (ou sources).

L'intégrité doit donc être vérifiée tout au long de la chaîne de calculs pour garantir la correction des résultats. Si les variables d'entrées sont correctes (ce qui est le cas par hypothèse puisqu'elles sont émises par l'environnement) et que tous les calculs effectués par la suite le sont également, alors le résultat l'est aussi. Les sources communiquées par l'environnement étant correctes par hypothèse, il suffit de s'assurer que tout résultat d'application d'une fonction dans l'arbre de calcul est aussi correct.

Les cas à analyser sont ceux de chaque acteur pour chaque équation dont l'intégrité doit être assurée (l'assurance d'un même calcul peut être acquise par différents moyens pour différents acteurs). À cette fin, le concepteur détermine la manière dont chaque acteur intéressé par la correction d'une variable (telle que définie dans les exigences d'intégrité) peut se convaincre de cette correction. Ce choix, qui se fait en sélectionnant le type de confiance à retenir par l'acteur en question pour chacune des fonctions, dépend des exigences et des contraintes imposées (certains contextes ne permettent pas la mise en œuvre de certains types de confiance). Il doit être effectué pour tous les acteurs concernés par les exigences d'intégrité. Deux acteurs peuvent donc être convaincus par un type de confiance différent de l'intégrité d'un même résultat.

Nous distinguons quatre types de confiance principaux : confiance par le calcul, par la sécurité, par la redevabilité ou par attestation. Un critère de choix déterminant est donc les relations existant entre les acteurs concernés et ce qu'ils sont prêts à accepter en matière d'incertitude sur la correction des valeurs, de contraintes sur l'architecture ou encore de coûts prévisibles.

Le tableau 3.7 récapitule les principaux critères de choix pour les différents types de confiance que nous présentons de manière plus détaillée dans le reste de cette section. Ces critères sont à mettre en regard de la spécification non fonctionnelle qui dépend du contexte.

Le motif de la confiance est la raison qui justifie cette confiance. La contrainte induite est la contrepartie architecturale de l'utilisation du type de confiance considéré. Les risques sont les cas dans lesquels l'intégrité d'un résultat pourrait être mise en échec sans que l'entité confiante puisse le savoir. Enfin, les coûts recensent les contreparties qui ne relèvent pas de la conception de l'architecture à ce niveau du développement. La prise en compte de l'ensemble de ces critères permet au concepteur d'effectuer le choix le plus adapté.

Confiance par le calcul. La confiance par le calcul découle du fait qu'un acteur qui effectue lui-même un calcul simple est assuré de la bonne exécution de ce calcul.

Dans ce cadre, ce type de confiance est le plus naturel mais il ne peut être mis en œuvre que quand l'acteur en question peut obtenir toutes les données d'entrée du calcul.

Cette possibilité est exclue quand les exigences en matière de vie privée lui interdisent l'accès à certaines données d'entrée.

Dans tous les cas, ce type de confiance est exclu si l'acteur ne possède pas les capacités de calcul nécessaires. Il faut alors s'orienter vers un autre type de choix tels que ceux présentés dans la suite.

Type de confiance	**Motif de la confiance**	**Contrainte induite**	**Risques** (hors sécurité et sûreté de fonctionnement)	**Coûts**
Calcul	autonomie	calcul local	inexistant par hypothèse	calculs
Sécurité	inviolabilité	vérification	inexistant par hypothèse	calculs et communications
Redevabilité	représentativité dissuasion	divulgation partielle authentification	fraude indétectée	infrastructure d'échantillonnage à déployer et à exploiter
Attestation	légitimité	authentification	fraude indétectable	négligeable si schéma d'authentification déjà en place

Tableau 3.7 – Critères pour le choix du type de confiance.

Confiance par la sécurité. Quand un acteur ne calcule pas lui-même un résultat, il doit disposer de déclarations provenant d'autres acteurs sur leur correction ou participer à un calcul sécurisé multipartite. La confiance par la sécurité repose sur la vérification systématique de la validité des propriétés déclarées ou la collaboration au calcul. Il s'agit d'une confiance vérifiée *a priori*, c'est-à-dire avant d'accepter la véracité de la propriété. Aussi quand un acteur A reçoit une déclaration de propriété p associée à un argumentaire $A(p)$ de l'acteur B, A s'assure de la véracité de p en vérifiant $A(p)$ par ses propres moyens. De la même façon, le calcul sécurisé multipartite est une option qui peut permettre de mieux se conformer aux exigences de confidentialité.

Ce type de confiance repose donc en dernier ressort sur la fiabilité des protocoles de vérification des argumentaires que nous supposons parfaite par hypothèse.

Les argumentaires peuvent prendre plusieurs formes. Ils peuvent être dissociés de la variable, fournis séparément, ou bien avoir été générés concomitamment au calcul de la variable. La première possibilité fait appel au domaine des preuves, souvent dites à divulgation nulle de connaissance ; la seconde au domaine des calculs sécurisés multipartites.

Bien que les garanties offertes par ces techniques soient très fortes, elles restent coûteuses en terme de calculs et de communications entre les composants. Elles sont le plus souvent interactives comme les preuves à divulgation nulle de connaissance ou les calculs sécurisés multipartites mais des efforts de recherche ont abouti à l'obtention de techniques non interactives (voir [BFM88 ; Bei+14]).

Il peut s'avérer nécessaire de se reposer sur un des autres types de confiance suivants si ces contraintes sont trop fortes.

Confiance par la redevabilité. La confiance par la redevabilité repose sur la possibilité de contrôle ultérieur de la véracité d'une déclaration : il s'agit d'une confiance vérifiable *a posteriori*, c'est-à-dire que la véracité de la déclaration est supposée jusqu'à preuve du contraire [5].

Les valeurs contrôlées correspondent généralement à un échantillon des valeurs sur lesquelles portent les déclarations et peuvent être recoupées avec d'autres sources d'information. Les vérifications sur place [6], les contrôles sporadiques et la confidentialité révocable en cas de litige sont des exemples d'application de ce type de confiance. Il s'agit d'autoriser la levée de la confidentialité des valeurs en cas de contradiction entre acteurs. La confiance est ici accordée *a priori* et une possibilité de contrôle existe pour l'acteur intéressé.

Les valeurs sur lesquelles les contrôles de cohérence ou de correction sont effectués sont choisies librement par l'acteur qui accorde sa confiance.

Les contrôles inopinés menés dans le cadre de ce type de confiance peuvent nécessiter des opérations de vérification supplémentaires et conduisent à des divulgations additionnelles de données (les échantillons collectés). Leur mise en œuvre représente donc un affaiblissement de

5. Il s'agit du type correspondant à ce que nous avons nommé « défis » dans la Section 2.4.
6. *Spotchecks* en anglais.

la protection des données personnelles de certains acteurs du système qui peut être incompatible avec certaines exigences.

Ce type de confiance n'apporte pas de garantie absolue : un acteur « chanceux » peut frauder régulièrement et ne jamais être confondu. Une composante essentielle de ce type de confiance est l'aspect dissuasif pour l'acteur contrôlé. Cette démarche se rapproche du principe de redevabilité où la confiance est établie *a priori* sous réserve de contrôles ultérieurs.

La mise en place de ce type de confiance a des impacts importants puisque les opérations de contrôle et le recueil additionnel des données doivent être prévus et intégrés dans l'architecture.

Ce type de confiance peut être utilisé par un acteur aussi bien pour vérifier que des sources ont bien été mesurées par d'autres acteurs que pour vérifier que des calculs ont été effectués correctement.

Lorsqu'aucun des types de confiance des paragraphes précédents ne peut être utilisé, il reste la possibilité d'opter pour la confiance par attestation.

Confiance par attestation. Nous appelons confiance par attestation celle qui repose sur la seule déclaration d'un tiers : l'acteur B qui fait confiance par attestation à l'acteur A considère qu'une propriété P attestée par A est effectivement vraie [7]. La confiance par attestation repose sur la confiance accordée en la simple déclaration d'un tiers. Ainsi la confiance par attestation peut être vue comme un dual de la confiance par sécurité (où peu importe l'acteur à l'origine du message, seule son argumentation est prise en compte pour ce qui est de la confiance dans la correction de la valeur associée). Il s'agit aussi d'un type de confiance *a priori*, comme la confiance par redevabilité. Elle peut cependant paraître plus faible que cette dernière en ce qu'aucune procédure n'est prévue pour la vérifier, même partiellement.

Le choix de ce type de confiance se justifie soit par la volonté de ne pas mettre en œuvre les moyens pour s'assurer effectivement de la correction, soit par l'impossibilité de le faire. Il ne reste alors qu'à déléguer complètement cette tâche à un tiers « de confiance ». Il est essentiel alors de pouvoir identifier avec certitude la source de l'attestation. C'est pourquoi l'identité de la source de la donnée doit être authentifiée.

Ce type de confiance est indiqué dans trois types de situation :

— pour les interactions avec l'environnement : il est nécessaire de disposer d'une source de confiance (sans quoi une garantie ne peut jamais être obtenue) ;
— pour traiter le cas de composants contrôlés par l'acteur concerné (composant sécurisé) ;
— pour intégrer des tiers de confiance dans une solution.

La mise en place de ce type de confiance nécessite donc une autorité de certification ou un système de signatures pour authentifier un message.

Si cela n'est pas possible en raison du contexte, ou parce que ce type de confiance n'est pas acceptable pour l'opération considérée, alors le concepteur peut revenir à l'étape de la localisation des calculs (voir Section 6.3.2.1) pour tenter une autre localisation. En effet, les

7. Il s'agit du type correspondant à ce que nous avons nommé « signature » dans la Section 2.4.

types de confiance dépendent fortement de la relation entre le composant où a lieu le calcul et l'acteur intéressé par la correction de ce calcul. La modification du premier entraîne de nouvelles possibilités pour le second. En cas d'impossibilité de trouver une solution convenable, il reste possible d'ajouter un composant dans le système (voir Section 3.5.1.1). Il faut alors définir l'acteur qui le contrôle, éventuellement un tiers de confiance, en ajoutant ce nouvel acteur au système (voir Section 3.4.1) et les données auxquelles l'accès lui est autorisé (les autres lui étant interdites par défaut, voir Section 3.4.3). La localisation du calcul ainsi fixée sur ce nouveau composant, il reste à déterminer un type de confiance approprié comme détaillé dans cette section.

Exemple. Il est maintenant possible de choisir un type de confiance pour chacune des opérations S, F et $\sum_t$ de l'architecture. Ce choix doit s'effectuer pour les exigences d'intégrité de chaque type d'acteur concerné : l'usager et le fournisseur. Les compteurs n'ont pas d'exigence d'intégrité sur les données et ne sont donc plus mentionnés dans le reste de cette étape.

La confiance par attestation est le type de confiance choisi classiquement lorsqu'un compteur est chargé d'effectuer une mesure, le compteur étant supposé être un composant métrologique sécurisé et certifié.

Nous considérons d'abord les exigences d'intégrité du fournisseur sur le résultat final *fee* du calcul. Le fournisseur est assuré de la bonne mesure S par une confiance par attestation de la part du compteur. La confiance par le calcul est exclue pour ce qui est de l'opération F car celle-ci est localisée chez un composant contrôlé par l'usager U. Le concepteur décide plutôt de retenir une confiance par attestation, prévoyant par exemple que le composant C_U est un module de plateforme de confiance. Pour l'opération $\sum_t$, la confiance par attestation est aussi sélectionnée de la même façon que pour l'opération F. Le fournisseur doit donc avoir confiance dans les attestations de tous les autres composants de l'architecture et être capable de vérifier leur authenticité.

Pour ce qui est de l'usager U, celui-ci est assuré de l'intégrité de la mesure S par le même moyen que le fournisseur : il établit une confiance par attestation dans le composant C_M. En revanche, pour ce qui est de la confiance en l'intégrité des calculs de F et de $\sum_t$, la confiance par le calcul est retenue puisqu'il a été choisi d'y localiser ces calculs.

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 3.8.

Calcul	Confiance de U	Confiance de P
$fee = \sum_t (price_t)$	Calcul	Attestation
$price_t = \mathrm{F}(cons_t)$	Calcul	Attestation
$cons_t = \mathrm{S}(ECONS_t)$	Attestation	Attestation

Tableau 3.8 – Types de confiance pour Ω_1.

3.5.3 Finalisation

Quand toutes les opérations ont été localisées et que toutes les exigences d'intégrité ont été satisfaites en s'appuyant sur un type de confiance défini, il devient possible de finaliser l'architecture. La finalisation de l'architecture consiste à intégrer les canaux de communication nécessaires. La Figure 3.9 présente ces étapes dans le cycle de conception.

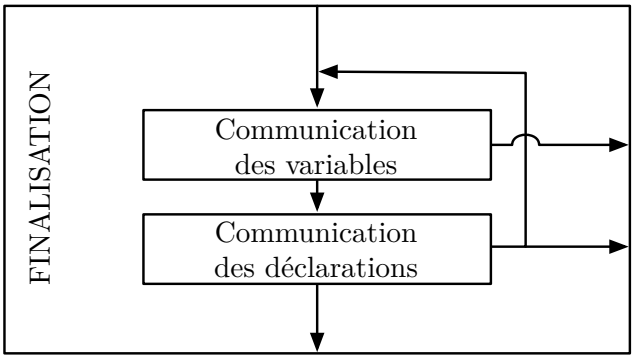

Figure 3.9 – Détail du cycle de finalisation de la Figure 3.5.

Les canaux de communication concernant les variables sont détaillés dans la Section 3.5.3.1 et les canaux concernant les déclarations le sont dans la Section 3.5.3.2.

3.5.3.1 Communication des variables

Le principe de minimisation implique qu'une donnée n'est communiquée que lorsqu'elle représente un argument d'un calcul simple (voir Section 6.3.2.1) ou pour satisfaire une exigence d'autorisation d'accès (voir Section 3.4.3). Les communications sont donc complètement déterminées par les calculs et par la spécification.

La démarche la plus efficace consiste à appliquer les opérations suivantes (dans cet ordre) :

1. ajouter les canaux de communications nécessaires justifiés par les nécessités de calculs ;
2. intégrer les canaux de communication nécessaires à la satisfaction des accès autorisés.

Si un canal doit assurer l'anonymat de l'expéditeur, il est nécessaire de revenir vers l'étape de structuration correspondante (voir Section 3.5.1.3).

Exemple. Un canal de communication doit être ajouté entre le composant du compteur C_{M} et le composant de l'usager C_{U} pour les variables $cons_t$ (afin que le calcul de F puisse s'effectuer). Un autre canal doit être rajouté de C_{U} vers le composant du fournisseur C_{P} pour qu'il puisse avoir connaissance de la variable *fee*.

3.5.3.2 Communication des déclarations

De la même façon que pour les variables, des déclarations doivent être communiquées entre composants pour satisfaire les exigences. Les déclarations regroupent communément les at-

testations et les preuves. Elles dépendent des types de confiance choisis précédemment (voir Section 3.5.2.3).

Exemple. Le compteur envoie une attestation de l'intégrité de la mesure $cons_t = \mathrm{S}(ECONS_t)$ à l'usager qui la fait suivre au fournisseur. L'usager envoie lui-aussi une attestation au fournisseur pour le reste des calculs des consommations finales $price_t = \mathrm{F}(cons_t)$ et $fee = \sum_t (price_t)$.

Cette étape constitue la dernière de la conception de l'architecture. La Figure 3.10 montre une représentation informelle de l'architecture obtenue. Les flèches en pointillés représentent les liens de confiance par attestation. Les flèches pleines représentent les envois d'un composant à un autre. Ceux-ci peuvent consister en des variables, des attestations ou des preuves (pour l'attestation par sécurité) dans cet exemple. Enfin, les équations dans les composants représentent la localisation des calculs.

3.6 Vérification

Les phases précédentes permettent de construire de façon raisonnée une (ou des) architecture(s) en appliquant des heuristiques reflétant les bonnes pratiques en matières de *PbD*. Ces heuristiques permettent de prendre en compte de manière systématique les exigences qui pèsent sur le système mais n'apportent pas à elles seules de garanties absolues de conformité. La dernière phase, la vérification des architectures obtenues, est donc nécessaire. Cette vérification peut être réalisée de manière plus ou moins formelle. Nous décrivons ici de manière informelle les vérifications à effectuer, la formalisation étant présentée dans le chapitre suivant.

Dernière phase du cycle de conception, la vérification de l'architecture n'en est pas moins une étape importante. La phase de vérification porte sur trois types de propriétés : la cohérence de l'architecture (Section 3.6.1), la satisfaction des exigences (Section 3.6.2) et la satisfaction des contraintes (Section 3.6.3), comme indiqué dans la Figure 3.11.

3.6.1 Cohérence de l'architecture

L'architecture doit être cohérente. Le concepteur s'assure à cette étape que l'ensemble des choix faits sont compatibles entre eux. La cohérence comprend notamment les propriétés suivantes :

— chaque donnée n'est définie (à travers un calcul ou une mesure) qu'une seule fois. Dans le cas contraire, la méthode de résolution est de créer une nouvelle variable et de lui appliquer la fonction appliquée deux fois en revenant à l'étape de la modélisation du service (voir Section 3.4.2). Par exemple, si deux composants étaient chargés de calculer $y = \mathrm{F}(x)$ alors le concepteur introduit la variable y' et rajoute l'équations $y = \mathrm{F}(x)$;
— chaque composant ne reçoit une donnée qu'une seule fois. Dans le cas contraire, la méthode de résolution est d'enlever un des canaux de communication en trop. De bons candidats sont les canaux dont l'élimination enlèverait toute justification pour le composant expéditeur d'avoir accès à la variable concernée. Par exemple, si le composant A reçoit x à la fois de B et de C qui le reçoit lui-même de B, alors les meilleurs canaux de communication à enlever sont ceux entre C et A et entre B et C à condition que

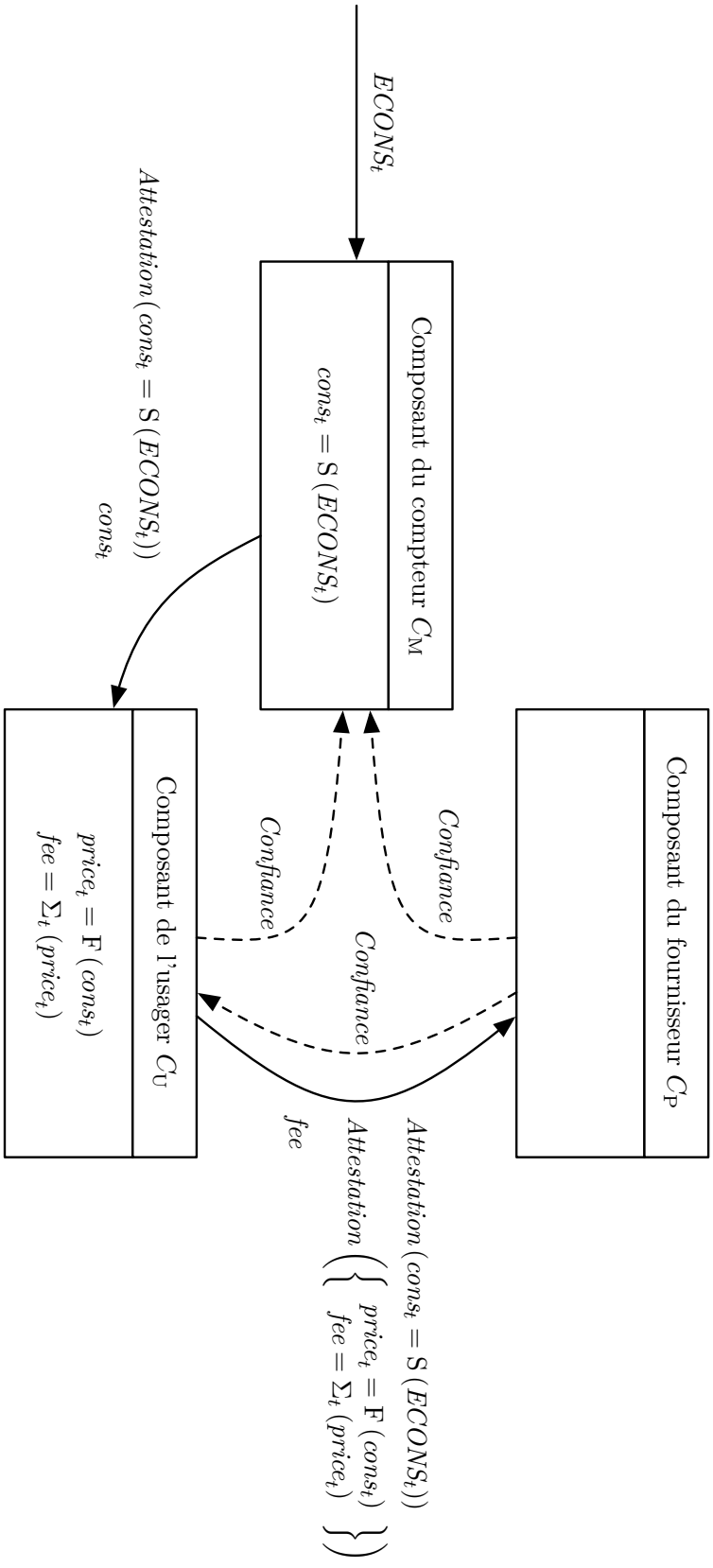

Figure 3.10 – Représentation informelle de l'architecture pour Ω_1.

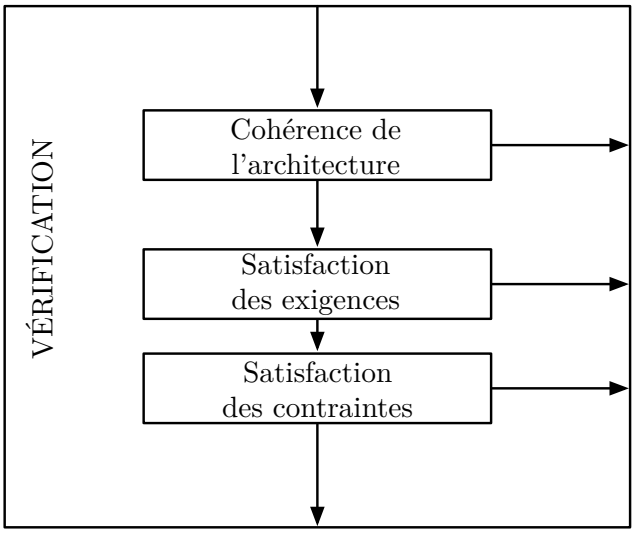

FIGURE 3.11 – Détail du cycle de vérification de la Figure 3.1.

C n'utilise pas la variable à d'autre fin. Dans le cas contraire, il peut être préférable d'enlever le canal entre B et A ;

— chaque composant effectuant un calcul simple a accès aux arguments nécessaires (soit parce qu'il les reçoit d'un autre composant, soit parce qu'il les calcule). Dans le cas contraire, la méthode de résolution consiste à rajouter les canaux de communication nécessaires en revenant à l'étape correspondante dans le cycle de développement (voir Section 3.5.3) ;

— chaque argument nécessaire à un calcul sécurisé multipartite doit être connu par un des participants au calcul. Si ce n'est pas le cas, la méthode de résolution est de donner accès à ces données à un des composants en revenant à l'étape correspondante dans le cycle de développement (voir Section 3.5.3).

Lorsque l'architecture est cohérente, la satisfaction des exigences qu'elle garantit peut alors être vérifiée.

Exemple. L'architecture conçue est cohérente au regard des quatre critères de cohérence énoncés. Il suffit pour s'en convaincre de considérer chaque critère un par un, et de vérifier qu'il est bien rempli. Les canaux de communication permettent aux calculs d'être effectuées aux localisations choisies (telles que décrites dans le Tableau 3.5) et chaque variable n'est ainsi calculée qu'une seule fois.

3.6.2 Satisfaction des exigences

À cette étape, il s'agit de confronter l'architecture conçue (suivant la démarche décrite en Section 3.5) à sa spécification (élaborée en suivant la démarche de la Section 3.4).

La satisfaction de chaque exigence doit être justifiée par un élément architectural (opération,

communication ou type de confiance). Il s'agit d'éléments exprimés explicitement sauf dans le cas de l'interdiction d'accès pour un acteur à des données. Dans ce cas, l'absence d'un canal de communication suffit à la justification à condition que cette donnée ne puisse être déduite d'un autre moyen comme le calcul d'une fonction dont les paramètres sont connus ou l'inversion d'une fonction dont le résultat est connu.

Exemple. Nous rappelons ici les exigences de confidentialité dans le Tableau 3.9 et d'intégrité dans le Tableau 3.10 après les itérations parcourues au cours de l'exemple.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	U	P
$cons_t$	U	P
$price_t$	U	P
fee	U, P	

Tableau 3.9 – Spécification des exigences de confidentialité pour Ω_1.

Les accès autorisés sont garantis par les canaux de communication ajoutés lors de l'étape de finalisation dans la Section 3.5.3. En ce qui concerne les interdictions d'accès, le fournisseur P ne peut déduire des *fee* les détails des consommations car la fonction $\sum_t$ n'est pas inversible en raison de son caractère agrégatif.

Relations	Connaissance
$fee = \sum_t (price_t)$	U, P
$price_t = \mathrm{F}(cons_t)$	U, P
$cons_t = \mathrm{S}(ECONS_t)$	U, P

Tableau 3.10 – Spécifications des exigences d'intégrité pour Ω_1.

Les types de confiance sélectionnés (tels que rappelés dans le Tableau 3.8) permettent aux exigences d'intégrité d'être satisfaites.

Cette satisfaction est une condition indispensable qui doit être atteinte par une architecture. Cette dernière doit aussi satisfaire les contraintes non fonctionnelles comme expliqué dans la section qui suit.

3.6.3 Satisfaction des contraintes

La satisfaction des contraintes fait écho à l'étape recensant les choix imposés au concepteur (voir Section 3.5.1.2). Elle est présentée ici comme l'étape finale de la vérification de l'architecture mais elle correspond en pratique à des contraintes qui doivent être prises en compte par le concepteur à chaque étape de la méthode. Il s'agit de s'assurer que les choix architecturaux sont acceptables.

La satisfaction des contraintes consiste en la revue de l'architecture en terme de performances, de déployabilité, d'utilisabilité ou encore de coûts. Ces critères non fonctionnels sont déterminés

par le contexte d'application. Si le système ne satisfaisait pas ces contraintes, il ne serait pas viable et il est donc nécessaire d'écarter tout choix incompatible. Cette vérification fait intervenir la connaissance du métier du concepteur et peut donner lieu à discussions avec le donneur d'ordres.

Exemple. La solution architecturale proposée dans l'exemple doit s'analyser dans son contexte d'application qui est la facturation de la consommation d'électricité basée sur des compteurs intelligents. Ces compteurs ne sont pas mobiles et un lien de communication fiable peut donc être établi avec ceux-ci. De plus, leurs capacités de calcul sont réduites au minimum puisqu'ils ne sont chargés que des mesures. Les calculs sont ainsi déportés dans une unité sous le contrôle de l'utilisateur, ce qui explique la confiance par sécurité déployée pour convaincre le fournisseur de l'intégrité du résultat.

Cependant, cette architecture n'est pas commune. Elle pose donc la question de son acceptabilité par les usagers en fonction de l'ergonomie du système final et des fournisseurs qui doivent accepter de perdre la main sur une partie de l'architecture qui leur était acquise jusqu'ici et de faire confiance à de nouvelles unités logiques.

3.7 Conclusion

Une des principales difficultés à laquelle fait face le concepteur durant le cycle de développement réside dans la manière de se repérer dans l'espace de conception. En effet, la combinatoire générée par toutes les possibilités de calcul et de communication entre divers composants aboutit rapidement à un ensemble dans lequel il devient très difficile de trouver des solutions. La démarche présentée précédemment guide le concepteur pas à pas à travers une série de questions pour dériver une architecture cohérente et satisfaisant toutes les exigences et contraintes. Afin de faciliter la tâche du concepteur, cette démarche a été mise en œuvre dans l'outil *CAPRIV* qui est présenté dans le Chapitre 5.

Les architectures ont cependant été décrites jusqu'à présent de manière purement informelle. Nous n'obtenons donc pas de garanties fortes à cette étape que les architectures dérivées respectent réellement les exigences de confidentialité et d'intégrité. Une manière de renforcer ces garanties est de fournir un cadre pour définir les architectures et pour raisonner sur leurs propriétés. Ce cadre fait l'object du chapitre suivant.

CHAPITRE 4

Cadre formel

4.1 Introduction

Les stratégies de conception présentées dans le chapitre précédent permettent de construire des architectures de manière systématique en tenant compte de toutes les contraintes et exigences décrites par le concepteur. Cependant, nous ne disposons pas à ce stade de preuve formelle de correction, notamment en matière de respect de la vie privée.

En pratique, les architectures sont souvent décrites soit de manière picturale en utilisant diverses sortes de graphiques munis de légendes définissant la signification des nœuds et des sommets, soit à l'aide de représentations semi-formelles telles que les diagrammes UML [BRJ05] (définissant des diagrammes de cas d'usage, de classes, de séquences, . . .). Même si ces représentations peuvent être très utiles (particulièrement quand elles sont standardisées), raisonner sur des exigences de vie privée est si complexe et subtil qu'on gagnerait à utiliser dans ce contexte des langages d'architecture définis formellement. En effet, un langage défini dans une logique mathématique qui permet un raisonnement sur les architectures étudiées devrait s'appuyer sur un système de preuve ou de vérification.

Raisonner sur les propriétés de vie privée est complexe pour différentes raisons. Le respect de la vie privée est une notion qui comporte de multiples facettes issues de principes qui ne sont pas eux-mêmes définis de manière précise. Dans ce contexte, il est donc nécessaire de circonscrire le problème et de définir précisément les aspects relatifs au respect de la vie privée qui sont pris en compte et transparaissent dans les choix de conception. Le fait que tous les aspects relatifs au respect de la vie privée ne soient pas susceptibles d'être formalisés n'est pas un obstacle en soi pour l'utilisation de méthodes formelles dans ce contexte : il faut simplement se focaliser sur les aspects du respect de la vie privée qui peuvent donner lieu à une formalisation et qui impliquent des raisonnements complexes, comme le principe de minimisation par exemple. Une autre source de complexité dans ce contexte est le fait que le respect de la vie privée est souvent vu comme entrant en conflit avec d'autres exigences comme par exemple les garanties d'authenticité, de correction, d'efficacité, ou d'utilisabilité.

Pour résumer, les méthodes formelles devraient dans ce contexte :

1. Permettre de définir précisément les concepts manipulés (exigences, hypothèses, garanties, ...).
2. Aider les concepteurs à explorer l'espace de conception et à raisonner sur les choix possibles. En effet, le respect de la vie privée dès la conception est souvent une affaire de choix [LM10] : de multiples options sont généralement disponibles pour réaliser un ensemble de fonctionnalités, certaines d'entre elles étant respectueuses de la vie privée, d'autres moins. Un défi majeur pour le concepteur est de comprendre toutes ces options, leurs forces et leurs faiblesses.
3. Fournir une base de documentation et des justifications rigoureuses pour les choix de conception effectués, ce qui va dans le sens des exigences de redevabilité. Cette dernière est définie dans le projet de règlement européen comme l'obligation pour le contrôleur de données « d'adopter des politiques appropriées et d'implémenter des mesures techniques et organisationnelles appropriées et démontrables pour assurer et être capable de démontrer de manière transparente que le traitement des données personnelles est effectué conformément à ce Règlement » [1].

Pour répondre à ces besoins de manière rigoureuse, nous proposons dans ce chapitre un cadre permettant de transcrire les architectures obtenues dans un formalisme qui permet la vérification de leurs propriétés plutôt que leur simple vérification informelle. Ce cadre repose sur les éléments architecturaux introduits précédemment qui sont définis formellement comme des primitives munies d'une sémantique à base de traces.

Nous présentons successivement les définitions des primitives architecturales dans la Section 4.2, des événements et des traces compatibles avec une architecture en Section 4.3 et des propriétés de respect de la vie privée et d'intégrité des architectures dans la Section 4.4.

4.2 Primitives architecturales

Des primitives architecturales ont été introduites dans le chapitre précédent pour permettre à un concepteur de construire des architectures. Ces primitives sont définies de manière formelle en Section 4.2.2 après l'introduction en Section 4.2.1 du langage des termes sur lequel elles reposent. La cohérence d'une architecture fait l'objet de la Section 4.2.3.

4.2.1 Langage des termes

Le langage des termes est utilisé pour exprimer le service du système à concevoir. Celui-ci est composé d'un ensemble d'équations $\Omega = \left\{\tilde{X} = T\right\}$ avec T les termes définis dans le Tableau 4.1. Il permet d'exprimer le modèle du service introduit (voir Section 3.4.2).

Le symbôle $\tilde{X}$ représente une variable (potentiellement indexée), X une variable ($X \in Var$), k une variable d'index ($k \in Index$), Cx une constante simple ($Cx \in Const$) et F une fonction ($F \in Fun$). La notation $\odot \mathrm{F}(X)$ désigne l'application itérative de la fonction F aux éléments du

1. « *The controller shall adopt appropriate policies and implement appropriate and demonstrable technical and organisational measures to ensure and be able to demonstrate in a transparent manner that the processing of personal data is performed in compliance with this Regulation* » dans le texte.

$$T ::= \tilde{X} \mid Cx \mid \mathrm{F}(T') \mid \odot F(T')$$
$$\tilde{X} ::= X \mid X_k$$

Tableau 4.1 – Langage des termes T de l'architecture.

tableau X (*eg.* la somme des éléments de X si F est la fonction + interprétée comme l'addition). L'ensemble *Fun* ne contient que des fonctions qui opèrent sur des valeurs de type entier (et non sur des tableaux) et retournent des entiers. Nous nous limitons à des fonctions F unaires pour l'application à des termes afin d'être en mesure de pouvoir garantir la complétude de l'axiomatique que nous présenterons dans la Section 4.4.2 [2]. Quand X représente un tableau, $Range\,(X)$ définit l'ensemble parcouru par son indice. Chaque variable X n'admet qu'un seul indice k qui peut lui être associé.

4.2.2 Langage des primitives architecturales

Les primitives architecturales sont construites à partir des termes précédents. Dans ce qui suit, nous utilisons Γ pour décrire l'ensemble des architectures respectant la syntaxe du langage $\mathcal{L}_{FPA}$ [3] présentée dans le Tableau 4.2.

$$\begin{array}{ll}
A ::= \{R\} & \\
R ::= Has_i\left(\tilde{X}\right) & \mid Receive_{i,j}\left(\{S\},\{\tilde{X}\}\right) \\
\quad \mid Compute_G\left(\tilde{X} = T\right) & \mid Check_i\left(\{Eq\}\right) \\
\quad \mid Verif_i^{Proof}\left(Pro\right) & \mid Verif_i^{Attest}\left(Att\right) \\
\quad \mid Spotcheck_{i,j}\left(X,\{Eq\}\right) & \mid Trust_{i,j}
\end{array}$$

$$\begin{array}{ll}
S ::= Pro \mid Att & Att ::= Attest_i\left(\{Eq\}\right) \\
Pro ::= Proof_i\left(\{P\}\right) & Eq ::= T_1\ Rel\ T_2 \\
P ::= Att \mid Eq & Rel ::= = \mid < \mid > \mid \leq \mid \geq
\end{array}$$

Tableau 4.2 – $\mathcal{L}_{FPA}$: langage des primitives architecturales.

La notation $\{Z\}$ est utilisée pour définir un ensemble d'expressions de catégorie Z, les indices i et j sont des index de composants et G est un ensemble de composants. Ceux-ci permettent de représenter les composants définis en Section 3.5.1.1 dans l'étape de structuration du cycle de développement. Les propriétés primitives *Eq* sont de simples équations et inéquations sur

2. Nous en expliquerons la raison après avoir prouvé la complétude de cette axiomatique.
3. *Formal Privacy Architecture Language*.

les termes T. Une architecture est un ensemble fini de relations R correspondant aux primitives suivantes :

$\boldsymbol{Has}_i(\tilde{\boldsymbol{X}})$ exprime le fait que la variable $\tilde{X}$ est une variable d'entrée pour le composant C_i (*eg.* captée ou mesurée). Il s'agit de la première primitive à utiliser dans l'approche ascendante décrite dans la phase d'opérationnalisation (voir Section 3.5.2) ;

$\boldsymbol{Receive}_{i,j}\left(\{\boldsymbol{S}\},\{\tilde{\boldsymbol{X}}\}\right)$ spécifie que le composant C_i peut recevoir du composant C_j des messages consistant en un ensemble de déclarations $\{S\}$ et un ensemble de variables $\{\tilde{X}\}$. Cette primitive correspond aux étapes des choix imposés et de finalisation de la démarche systématique (voir Sections 3.5.1.2 et 3.5.3). Les déclarations peuvent être des deux types suivants :

$\boldsymbol{Proof}_i$ $(\{\boldsymbol{P}\})$ pour la preuve d'un ensemble de propriétés P ;

$\boldsymbol{Attest}_i$ $(\{\boldsymbol{Eq}\})$ pour l'attestation d'un ensemble d'équations Eq, c'est-à-dire une simple déclaration par le composant C_i que ces équations sont vraies.

Ces deux déclarations dépendent des types de confiance choisis (par sécurité et par attestation respectivement) dans l'architecture (voir Section 3.5.2.3) ;

$\boldsymbol{Compute}_G\left(\tilde{\boldsymbol{X}}=\boldsymbol{T}\right)$ pour exprimer le fait qu'un ensemble de composants peut calculer la valeur à assigner à une variable définie par l'équation $\tilde{X} = T$ (il s'agit d'un calcul simple s'il n'y a qu'une entité dans G et nous écrivons $Compute_i$ pour simplifier, d'un calcul sécurisé multipartite s'il y en a plusieurs). Ces opérations sont introduites dans l'architecture à l'étape de la localisation des calculs (voir Section 6.3.2.1) et traduit une confiance par le calul ou la sécurité en fonction du nombre d'éléments dans G (voir Section 3.5.2.3) ;

$\boldsymbol{Check}_i$ $(\{\boldsymbol{Eq}\})$ pour exprimer le fait qu'un composant peut aussi vérifier qu'un ensemble de propriétés Eq sont vraies. Cette primitive est utilisée dans le cadre du type de confiance par le calcul (voir Section 3.5.2.3) ;

$\boldsymbol{Verif}_i^{\boldsymbol{Proof}}$ $(\boldsymbol{Pro})$ pour exprimer le fait qu'un composant peut vérifier la preuve d'une propriété Pro reçue d'un autre composant dans le cadre de la confiance par sécurité (voir voir Section 3.5.2.3) ;

$\boldsymbol{Verif}_i^{\boldsymbol{Attest}}$ $(\boldsymbol{Att})$ pour exprimer le fait qu'un composant peut vérifier l'origine d'une attestation Att reçue correspondant au type de confiance par attestation (voir Section 3.5.2.3) ;

$\boldsymbol{Spotcheck}_{i,j}$ $(\boldsymbol{X},\{\boldsymbol{Eq}\})$ pour exprimer le fait qu'un composant peut effectuer un contrôle sporadique. Ce contrôle consiste en une demande par un composant C_j d'une valeur X_k suivie de la vérification que cette valeur satisfait Eq. Cette primitive correspond au choix du type de confiance par redevabilité (voir Section 3.5.2.3) ;

$\boldsymbol{Trust}_{i,j}$ pour exprimer le fait qu'un composant C_i a confiance dans les déclarations d'un composant C_j. Cette primitive est à nouveau reliée au type de confiance par attestation (voir Section 3.5.2.3).

Par hypothèse, nous choisissons de ne travailler que sur des architectures finies ne comportant qu'un seul contrôle sporadique $Spotcheck_{i,j}(X,\{Eq\})$.

Notons que le langage proposé ici reflète l'ensemble des primitives (*PETs*) supposé disponible pour construire des architectures (preuves à divulgation nulle de connaissance, contrôles

sporadiques, attestations, . . .). Ce langage n'est pas fermé, il peut être étendu avec d'autres primitives en fonction des techniques disponibles dans des contextes particuliers (communications anonymes par exemple).

La relation Dep_i est introduite pour exprimer le fait que des variables peuvent être dérivées d'autres variables. Par exemple, $Dep_i\left(\tilde{Y}, \tilde{X}\right)$ signifie que la valeur de $\tilde{Y}$ peut être obtenue par C_i s'il possède la variable $\tilde{X}$. On aura par exemple $Dep_i\,(y, x)$ si $y = \mathrm{F}\,(x)$ appartient à Ω. Par contre, si $y = \mathrm{H}\,(x)$ avec H une fonction de hachage, l'absence d'une relation $Dep_i\,(x, y)$ empêche le composant C_i de dériver la valeur de x de celle de y, ce qui correspond à la propriété de masquage. Dep_i doit inclure notamment les relations de dépendance entre les variables résultats $\tilde{X}$ et la variable membre de T apparaissant dans les calculs $Compute_G\left(\tilde{X} = T\right)$.

4.2.3 Cohérence d'une architecture

Dans ce qui suit, nous ne considérons que des architectures cohérentes. Une architecture est dite cohérente si [4],avec i, j et k différents :

- (HC1) l'expression équationnelle du service Ω ne contient pas de cycle (ce qui est le cas si le service correspond à un graphe dirigé acyclique comme décrit en Section 3.4.2) ;
- (HC2) seules les variables d'entrée (qui n'apparaissent en partie gauche d'aucune équation du service) apparaissent en argument de Has_i ;
- (HC3) un ensemble de composants ne peut effectuer un calcul ($Compute_G(\tilde{X} = T)$) que si la variable apparaissant dans T peut être mesurée (Has_i), calculée ($Compute_H$), ou bien reçue ($Receive_{i,j}$) par un composant $i \in G$ (et $i \in H$) et si $Dep_i\left(\tilde{X}, \tilde{Y}\right)$ avec $\tilde{Y}$ la variable apparaissant dans T ;
- (HC4) un composant ne peut effectuer une vérification ($Check_i\,(Eq)$) que s'il peut mesurer (Has_i), participer au calcul de ($Compute_G$ avec $i \in G$) ou recevoir ($Receive_{i,j}$) chaque variable apparaissant dans Eq ;
- (HC5) un composant ne peut envoyer une variable ($Receive_{i,j}$) que s'il peut la mesurer (Has_j), participer à son calcul ($Compute_G$ avec $j \in G$) ou la recevoir ($Receive_{j,k}$) ;
- (HC6) chaque variable ne peut être mesurée (Has_i) ou calculée ($Compute_G$) qu'une seule fois dans l'architecture ;
- (HC7) chaque composant ne peut que (exclusivement) mesurer (Has_i), participer au calcul de ($Compute_G$ avec $i \in G$), ou recevoir ($Receive_{i,j}$) une variable ;
- (HC8) un composant ne peut envoyer que des attestations ($Attest_k\,(\{Eq\})$) dont il est l'auteur ($Receive_{i,k}\,(\{Attest_k\,(\{Eq\})\}, \{X\})$) ou qu'il peut lui-même recevoir ($Receive_{i,j}\,(\{Attest_k\,(\{Eq\})\}, \{X\})$) ;
- (HC9) un composant ne peut vérifier ($Verif_i$) que des propriétés qu'il peut recevoir ($Receive_{i,j}$) ;
- (HC10) un composant ne peut effectuer un contrôle sporadique ($Spotcheck_{i,j}\,(X, \{Eq\})$) que si X_k est une variable d'entrée, si toutes les équations de $\{Eq\}$ ont les autres variables que X_k qui peuvent être mesurées (Has_i), calculées ($Compute_G$ avec $i \in G$), reçues ($Receive_{i,j}$) ou qui sont liées dans $\{Eq\}$ (et sont alors discrètes) et si X_k ne peut

4. Nous allégeons la notation en ne retenant pour les primitives architecturales que les parties syntaxiques nécessaires à la compréhension sans leurs paramètres.

par ailleurs être ni mesurée, ni calculée, ni reçue par ce composant. Enfin, il faut que $Dep_i\left(\tilde{X},\tilde{Y}\right)$ pour toutes les Eq faisant intervenir $\tilde{X}$ en partie gauche et $\tilde{Y}$ en partie droite. Un composant a la capacité de mesurer, de calculer, de recevoir, de vérifier, d'attester, de prouver ou d'effectuer un contrôle sporadique si la primitive architecturale correspondante dans $\mathcal{L}_{FPA}$ est présente dans l'architecture. Une architecture cohérente induit donc un flot d'informations bien défini.

Concernant la relation Dep_i, la contrainte de cohérence (HCDep) qui lui est associée est que si $Dep_i\left(\tilde{X},\tilde{Y}\right)$ alors il existe une fonction F totale telle que $\mathrm{F}\left(\tilde{Y}\right) = \tilde{X}$ avec $\tilde{X}$ complètement définie (différente de $\perp$ ou dont toutes les valeurs sont différentes de $\perp$ dans le cas d'un tableau), ce qui traduit le fait qu'il est dans tous les cas possible de construire $\tilde{X}$ à partir de $\tilde{Y}$.

Les hypothèses de cohérence (HC1), (HC3-5) et (HC8-10) permettent d'établir un ordre partiel, correspondant à un service effectivement calculable, que nous notons $<<^A$ sur l'ensemble des primitives architecturales de l'architecture A. Par exemple, l'hypothèse de cohérence (HC3) entraîne les relations suivantes dans l'ordre $<<^A$:

— $Has_i\left(\tilde{Y}\right) <<^A Compute_G\left(\tilde{X} = T\right)$ si $i \in G$ et $\tilde{Y} \in Args\,(T)$;
— $Compute_H\left(\tilde{Y} = U\right) <<^A Compute_G\left(\tilde{X} = T\right)$ si $H \cap G \neq \emptyset$ et $\tilde{Y} \in Args\,(T)$;
— $Receive_{i,j}\left(\{S\},\{\tilde{Y}\}\right) <<^A Compute_G\left(\tilde{X} = T\right)$ si $i \in G$ et $\tilde{Y} \in Args\,(T)$.

Exemple. Nous poursuivons l'exemple utilisé dans le chapitre précédent. La Figure 4.1 représente l'architecture exprimée à l'aide de primitives formelles.

Les composants sont représentés par des boîtes. Les flèches pleines représentent les communications et les flèches en pointillés les liens de confiance par attestation entre entités [5].

Le compteur reçoit la variable d'entrée $ECONS_t$ et effectue la mesure par l'application de S. Cette mesure $cons_t$ est envoyée au composant de l'usager pour effectuer les calculs de tarification F et d'agrégation $\odot+$ pour déterminer le montant fee de la facture. Celui-ci est ensuite envoyé au composant du fournisseur avec une attestation sur la mesure S par C_{M} et une attestation sur la tarification F et l'agrégation $\odot+$ par le composant de l'usager C_{U}. La vérification de l'authenticité de ces attestations permet au composant du fournisseur, et enfin au fournisseur lui-même de recevoir le montant fee en étant convaincu de son intégrité (relativement aux $price_t$, $cons_t$ et $ECONS_t$).

4.3 Traces d'événements

La définition de la sémantique d'une architecture est basée sur un ensemble de traces qui lui sont compatibles. L'effet des événements constituant ces traces est modélisé par leurs conséquences sur les états des composants.

Le langage des traces est présenté en 4.3.1 avant la définition de leur cohérence en 4.3.2 et de leur compatibilité avec une architecture en 4.3.3. Enfin, la notion d'état des composants est introduite en 4.3.4 avant le développement de la sémantique des traces en 4.3.5.

5. Les flèches n'ont pas de sémantique : les primitives formelles suffisent à définir complètement les communications et les liens de confiance par attestation.

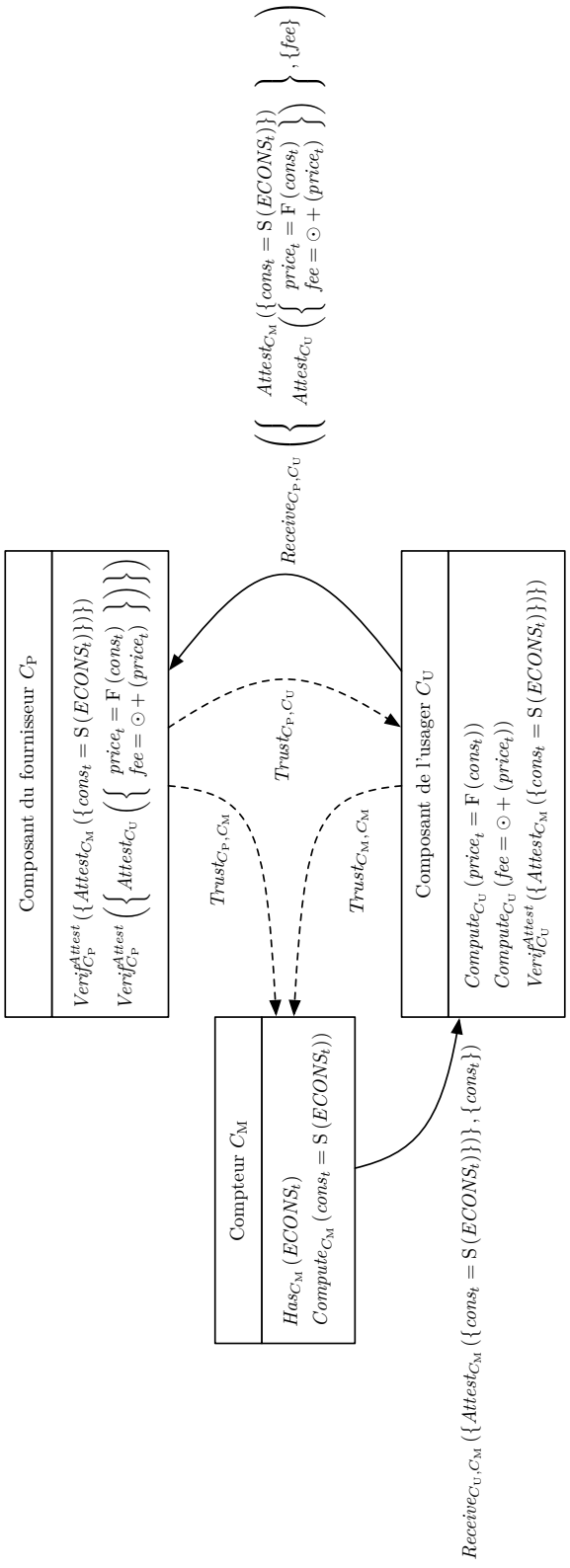

FIGURE 4.1 – Représentation dans le langage $\mathcal{L}_{FPA}$ de l'architecture pour Ω_1.

4.3.1 Langage des traces d'événements

Une trace est une séquence d'événements de haut-niveau se produisant dans le système. Il repose sur un langage de termes étendu T^{ϵ} par rapport au langage de termes T de l'architecture en permettant aux indices d'être instanciés par des constantes d'index $Ck \in \mathbb{N}$ comme défini dans le Tableau 4.3 [6]. Comme chaque variable X n'admet qu'un seul indice k, la dénomination X_{Ck} n'est pas ambiguë et désigne une valeur du tableau X. Les autres types $\tilde{X}$, Cx, F, et k correspondent à ceux de T définis précédemment.

$$
\begin{aligned}
T^{\epsilon} &::= \tilde{X} \mid Cx \mid F(T^{\epsilon\prime}) \mid \odot\mathrm{F}(X) \\
\tilde{X} &::= X \mid X_K \\
K &::= k \mid Ck
\end{aligned}
$$

Tableau 4.3 – Langage des termes T^{ϵ} des événements.

Le langage des traces est présenté dans le Tableau 4.4.

$$
\begin{aligned}
\theta &::= Seq(\epsilon) \\
\epsilon &::= Has_i\left(\tilde{X} : V\right) && \mid Receive_{i,j}\left(\{S\}, \{\tilde{X} : V\}\right) \\
&\quad \mid Compute_G\left(\tilde{X} = T^{\epsilon}\right) && \mid Check_i\left(\{Eq\}\right) \\
&\quad \mid Verif_i^{Proof}\left(Pro\right) && \mid Verif_i^{Attest}\left(Att\right) \\
&\quad \mid Spotcheck_{i,j}\left(X_{Ck} : V, \{Eq\}\right)
\end{aligned}
$$

Tableau 4.4 – Langage des événements ϵ et traces θ.

Les événements peuvent être vus comme des instantiations des primitives architecturales (à l'exception de la primitive $Trust_{i,j}$ qui correspond à une hypothèse plus qu'à un événement). Nous supposons que V est du même type que $\tilde{X}$ et correspond soit à une valeur dans $\mathbb{N}$, soit à un tableau $< v_1, \ldots, v_k >$ de valeurs dans $\mathbb{N}$ parcourant $Range(X)$. On a donc $V \in Val_{\perp}$ avec $Val_{\perp} = \mathbb{N} \cup \{\perp\} \cup \sum(\mathbb{N} \cup \{\perp\})$ (avec $\perp$ la valeur indéfinie) avec $\sum$ désignant l'ensemble des séquences. Par exemple, un événement $Receive_{i,j}\left(\{S\}, \{\tilde{X} : V\}\right)$ définit la valeur V d'une variable $\tilde{X}$ reçue par le composant C_i. De manière similaire, $Spotcheck_{i,j}\left(X_{Ck} : V, \{Eq\}\right)$ définit l'index spécifique Ck (membre de $Range\ (X)$) choisi par le composant C_i pour la vérification sporadique ainsi que la valeur V de X_{Ck}. Toutes les variables d'index apparaissant dans des événements, exceptées celles apparaissant dans les propriétés de $Receive_{i,j}$, $Verif_i^{Proof}$, $Verif_i^{Attest}$ et $Spotcheck_{i,j}$ doivent appartenir à $\mathbb{N}$. En d'autres termes, elles doivent être instanciées et désigner une valeur d'indice dans le tableau correspondant. Enfin, comme $Has_i\left(\tilde{X} : V\right)$ correspond à la mesure d'une

6. Dans la suite, nous désignerons sans distinction les deux langages par T lorsqu'il n'y aura pas d'ambiguité sur le langage des termes désigné.

valeur de l'environnement, nous supposons que V est complètement défini ($V \in Val$ avec $Val = \mathbb{N} \cup \sum(\mathbb{N})$). De la même façon, les réceptions et les calculs définissent complètement les variables. Enfin, dans le cas où $\tilde{X}$ est une variable tableau, nous supposons que les $v_1, \ldots, v_n$ sont tous indépendants les uns des autres de telle sorte que la connaissance d'une des éléments du tableau ne permette pas de retrouver la valeur d'un autre élément du tableau.

4.3.2 Cohérence d'une trace

Nous définissons la relation C telle que $C(\epsilon, \alpha)$ est satisfaite si et seulement si l'événement ϵ peut être obtenu d'une primitive architecturale α en spécifiant des valeurs V pour les variables et en instanciant les variables d'index k à des entiers Ck. Cette relation est appelée relation de correspondance.

Des hypothèses de cohérence similaires aux hypothèses de cohérence d'une architecture s'appliquent aux traces (voir Section 4.2.3) en remplaçant les primitives par les événements correspondants au sens de la relation de correspondance. Il s'y ajoute des hypothèses relatives à l'ordre des événements. Instinctivement, les variables ne peuvent être utilisées par un composant qu'après avoir été mesurées, calculées ou reçues. De même, les propriétés ne peuvent être vérifiées qu'après en avoir reçu une attestation ou une preuve.

De plus, en cas de contrôle sporadique, les événements de la trace peuvent utiliser la variable contrôlée X_{Ck} pour effectuer des calculs, des réceptions, des vérifications, des attestations et des preuves.

Ainsi, l'hypothèse de cohérence d'architecture (HC3) (voir Section 4.2.3) devient, pour les traces : (HC3') un événement $Compute_G\left(\tilde{X} = T\right)$ correspondant à un ensemble de composants effectuant un calcul ne peut apparaître dans la trace que si chaque variable apparaissant dans T a été préalablement (dans un événement d'indice inférieur dans la trace) mesurée (Has_i), calculée ($Compute_H$), reçue ($Receive_{i,j}$) ou bien contrôlée de manière sporadique ($Spotcheck_{i,j}$) par un des composant $i \in G$ (et $i \in H$) et si $Dep_i\left(\tilde{X}, \tilde{Y}\right)$ avec $\tilde{Y}$ la variable apparaissant dans T.

Nous obtenons ainsi des traces que nous appelons traces cohérentes. Nous nommons *Event* l'ensemble des événements ϵ et *Trace* l'ensemble des traces cohérentes θ.

De la même façon que pour l'architecture, les hypothèses de cohérence des traces permettent d'établir un ordre partiel que nous notons $<<^\theta$ sur l'ensemble des événements de la trace θ. Par exemple, l'hypothèse de cohérence (HC3') entraîne les relations suivantes dans l'ordre $<<^\theta$:

- $Has_i\left(\tilde{Y} : V\right) <<^\theta Compute_G\left(\tilde{X} = T\right)$ si $i \in G$ et $\tilde{Y} \in Args(T)$;
- $Compute_H\left(\tilde{Y} = U\right) <<^\theta Compute_G\left(\tilde{X} = T\right)$ si $H \cap G \neq \emptyset$ et $\tilde{Y} \in Args(T)$;
- $Receive_{i,j}\left(\{S\}, \{\tilde{Y} : V\}\right) <<^\theta Compute_G\left(\tilde{X} = T\right)$ si $i \in G$ et $\tilde{Y} \in Args(T)$;
- $Spotcheck_{i,j}\left(Y_{Ck} : V, \{Eq\}\right) <<^\theta Compute_G\left(\tilde{X} = T\right)$ si $i \in G$ et $Y_{Ck} \in Args(T)$.

Nous construisons les hypothèses de cohérence des traces à partir des hypothèses de cohérence de l'architecture. Par conséquent, pour toute architecture A et pour toute trace θ avec pour tout $\epsilon, \epsilon' \in \theta$ et $\alpha, \alpha' \in A$, si $C(\epsilon, \alpha)$ et $C(\epsilon', \alpha')$ et $\alpha <<^A \alpha'$, alors $\epsilon <<^\theta \epsilon'$. La relation d'ordre $<<^\theta$ est donc compatible avec $<<^A$ pour tous les éléments de θ et A correspondants.

Nous remarquons que les traces sont des séquences. Toute trace est donc munie d'un ordre total naturel. Les hypothèses de cohérence impliquent que cet ordre total est une extension linéaire de l'ordre partiel $<<^{\theta}$.

La cohérence des architectures et des traces ayant été définie, nous ne nous intéressons maintenant qu'à celles-ci. Nous pouvons donc établir le lien entre ces architectures et ces traces.

4.3.3 Compatibilité d'une trace avec une architecture

Nous définissons la notion de compatibilité d'une trace avec une architecture. Intuitivement, une trace est compatible avec une architecture si ses événements, hormis les $Compute_i$, correspondent à une primitive de cette architecture (avec une contrainte d'unicité pour les contrôles sporadiques $Spotcheck_{i,j}$).

Définition 1 (Compatibilité)**.** *Une trace θ de longueur $\overline{\theta}$ est compatible avec une architecture A si et seulement si :*

$$\begin{aligned}\forall a \in [1,\overline{\theta}],\ si\ \theta_a \neq Compute_i\left(\tilde{X}=T\right)\ alors\ \exists \alpha \in A, C(\theta_a,\alpha)\ et\\ si\ \theta_a = Spotcheck_{i,j}\left(X_{Ck}:V,\{Eq\}\right)\\ alors\ \forall b \in [1,\overline{\theta}], b \neq a \Rightarrow \forall Ck', V', Eq',\\ \theta_b \neq Spotcheck_{i,j}\left(X_{Ck'}:V',\{Eq'\}\right)\end{aligned}$$

avec $C(\epsilon,\alpha)$ la relation de correspondance définie précédemment.

La première condition dans la définition de la compatibilité établit que, excepté pour les événements correspondant à des calculs, seuls les événements qui sont des instances de primitives architecturales appartenant à l'architecture A peuvent apparaître dans la trace θ. Les événements de calcul sont exclus afin d'exprimer les actions à la portée d'un agent curieux essayant de dériver la valeur d'une variable $\tilde{X}$ en s'appuyant sur les valeurs des variables qu'il possède déjà. Par conséquent, les traces compatibles peuvent inclure des calculs qui ne sont pas prévus dans l'architecture, pourvu que le composant C_i puisse mesurer, calculer ou recevoir toutes les variables nécessaires pour effectuer ce calcul comme spécifié par la relation de dépendance Dep_i (par cohérence). Conformément aux hypothèses (voir Section 3.3), le modèle d'adversaire considéré ici permet les calculs de nouvelles variables, les calculs selon une fonction différente du modèle et donc la communication de valeurs incorrectes à d'autres composants [7]. La seconde condition exprime le fait que les vérifications sporadiques ne peuvent être effectuées qu'une seule fois. Cette condition pourrait être assouplie à travers l'introduction d'un seuil t comme paramètre supplémentaire pour exprimer le fait qu'au maximum t vérifications sporadiques peuvent être effectuées.

Nous notons $T(A)$ l'ensemble des traces compatibles d'une architecture A. La sémantique des événements est définie par l'effet de ces traces sur les états des composants. Nous définissons donc la notion d'état d'un composant avant de présenter la sémantique des traces d'événements.

7. Cependant, les attestations, preuves, vérifications et contrôles sporadiques ne peuvent par contre pas être incorrects.

4.3.4 État des composants

L'état d'un composant s'appuie sur les ensembles de variables et de propriétés :

$$
\begin{aligned}
State_{\perp} &= (State_V \times State_P \times State_P) \cup \{Error\} \\
State_V &= (Var \rightarrow Val_{\perp}) \\
State_P &= \left\{\{Eq\} \cup \left\{Trust_{i,j}\right\}\right\}
\end{aligned}
$$

L'état d'un composant est soit l'état d'erreur *Error*, soit un triplet constitué :

- d'un état des variables assignant une valeur, ou la valeur indéfinie $\perp$ [8], à chaque variable ;
- d'un premier état de propriétés définissant l'ensemble des propriétés connues par un composant ;
- d'un second état de propriétés définissant l'ensemble des propriétés crues par un composant (issues de vérifications sporadiques). Il est nécessaire de distinguer les propriétés connues des propriétés crues car elles correspondent d'un point de vue sémantique à deux modalités différentes.

Dans ce qui suit, nous utilisons σ pour désigner l'état global (état des composants $\langle C_1, \ldots, C_n \rangle$) défini sur $State_{\perp}^n$.

L'état initial pour l'architecture A est noté $Init^A = \langle Init_1^A, \ldots, Init_n^A \rangle$ avec :

$$
\forall i \in [1, n], Init_i^A = \left(Empty, \left\{Trust_{i,j} | Trust_{i,j} \in A\right\}, \emptyset\right)
$$

où *Empty* est l'état de variables indéfinies ($\forall X \in Var, Empty(X) = \perp$). La seule information contenue dans l'état initial est l'ensemble des propriétés de confiance spécifiées dans l'architecture.

4.3.5 Sémantique des traces d'événements

La fonction sémantique S_T est définie dans le Tableau 4.5. Elle spécifie l'impact d'une trace sur l'état de chaque composant C_i. Elle est définie comme une itération sur la trace. La fonction S_E définit l'impact de chaque type d'événement sur l'état des composants.

La notation $\epsilon.\theta$ est utilisée pour désigner une trace dont le premier élément est l'événement ϵ et dont le reste est θ. Chaque événement modifie seulement l'état du composant C_i (ou des composant $\{C_i\}$ en cas de calcul multipartite). La modification est exprimée par $\sigma[\sigma_i/(v, pk, pb)]$ (ou $\sigma[\sigma_i/Error]$ dans le cas de l'état indéfini) qui remplace les états des variables et des propriétés du composant C_i par v, pk, et pb respectivement. Nous supposons qu'il n'y a pas d'événement $\theta_{a'}$ avec $a' > a$ qui implique un composant C_i si son état σ_i est égal à *Error* après l'occurrence de θ_a (en d'autres termes, toute erreur dans l'exécution d'un composant entraîne son arrêt).

8. Nous utilisons $\perp$ pour dénoter les valeurs indéfinies, c'est à dire les valeurs qui n'ont pas été fixées, par opposition aux valeurs issues d'erreurs (comme une division par zéro ou des erreurs de type par exemple) qui ne sont pas prises en compte ici pour des raisons de simplification. Il suffirait pour ce faire d'introduire une valeur spécifique $\perp'$.

$$
\begin{aligned}
S_T &: \mathit{Trace} \times \mathit{State}_\perp^n \to \mathit{State}_\perp^n \\
S_E &: \mathit{Event} \times \mathit{State}_\perp^n \to \mathit{State}_\perp^n \\
S_T(\langle\rangle, \sigma) &= \sigma \\
S_T(\epsilon.\theta, \sigma) &= S_T(\theta, S_E(\epsilon, \sigma)) \\
S_E(\mathit{Has}_i\left(\tilde{X} : V\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v[\tilde{X}/V], \sigma_i^{pk}, \sigma_i^{pb})] \\
S_E(\mathit{Receive}_{i,j}\left(\{S\}, \{\tilde{X} : V\}\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v[\{\tilde{X}/V\}], \sigma_i^{pk}, \sigma_i^{pb})] \\
S_E(\mathit{Compute}_G\left(\tilde{X} = T\right), \sigma) &= \sigma[\{\sigma_i/(\sigma_i^v[\tilde{X}/\varepsilon(T, \underset{i\in G}{\cup}\sigma_i^v)], \sigma_i^{pk} \cup \{\tilde{X} = T\}, \sigma_i^{pb})\}] \\
&\qquad \text{pour les } i \in G \\
S_E(\mathit{Check}_i\left(E\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup E, \sigma_i^{pb})] \\
&\qquad \text{si } \forall Eq \in E, \varepsilon(Eq, \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ autrement} \\
S_E(\mathit{Verif}_i^{\mathit{Proof}}\left(\mathit{Proof}_j(E)\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup \{Eq | Eq \in E \text{ ou} \\
&\qquad\qquad (\mathit{Attest}_{j'}(E') \in E \text{ et} \\
&\qquad\qquad Eq \in E' \text{ et} \\
&\qquad\qquad \mathit{Trust}_{i,j'} \in \sigma_i^{pk})\}, \sigma_i^{pb})] \\
&\qquad \text{si } \overline{\mathit{Verif}}^{\mathit{Proof}}((E), \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ autrement} \\
S_E(\mathit{Verif}_i^{\mathit{Attest}}\left(\mathit{Attest}_j(E)\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup \{Eq | Eq \in E \text{ et} \\
&\qquad\qquad \mathit{Trust}_{i,j} \in \sigma_i^{pk}\}, \sigma_i^{pb})] \\
&\qquad \text{si } \overline{\mathit{Verif}}^{\mathit{Attest}}((E), \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ autrement} \\
S_E(\mathit{Spotcheck}_{i,j}\left(X_{Ck} : V, E\right), \sigma) &= \sigma[\sigma_i/(\sigma_i^v[X_{Ck}/V], \sigma_i^{pk}, \sigma_i^{pb} \cup E)] \\
&\qquad \text{si } \forall Eq \in E, \\
&\qquad\quad \varepsilon(Eq[\tilde{X}/X_{Ck}], \sigma_i^v[X_{Ck}/V]) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ autrement}
\end{aligned}
$$

Tableau 4.5 – Sémantique des traces d'événements S_T et S_E.

L'effet de Has_i et $\mathit{Receive}_{i,j}$ sur l'état des variables du composant C_i est le remplacement des valeurs des variables $\tilde{X}$ de T^ϵ (qui doivent être initialement égales à $\perp$ par l'hypothèse de cohérence des traces) par de nouvelles valeurs $V \in \mathit{Val}$, ce qui est noté $\sigma_i^v[\tilde{X}/V]$.

L'effet de $\mathit{Compute}_G(\tilde{X} = T)$ est d'assigner à la variable $\tilde{X}$ la valeur correspondant au résultat de l'évaluation de T dans l'état des variables courant σ_i^v de tous les composants C_i pour $i \in G$, ce qui est défini par $\varepsilon(T, \{\sigma_i^v\})$. Les valeurs de ces variables sont cohérentes et sans

ambiguïté car nous travaillons sur des traces cohérentes. L'événement de vérification sporadique $Spotcheck_{i,j}(X_{Ck} : V, E)$ assigne la valeur V à X_{Ck}. Les autres événements n'ont pas d'effet sur l'état des variables du composant C_i.

La plupart des événements ont aussi un effet sur les états des propriétés. Cet effet est l'ajout aux états correspondants des nouvelles connaissances et croyances issues des événements. Pour $Compute_i\left(\tilde{X} = T\right)$, cette nouvelle connaissance est l'égalité $\tilde{X} = T$; pour les événements $Check_i$, $Verif_i$ et $Spotcheck_{i,j}$, la nouvelle connaissance correspond aux propriétés vérifiées. Dans tous les cas, excepté pour $Spotcheck_{i,j}$, ces propriétés sont ajoutées à l'état des propriétés *pk* parce qu'elles sont admises comme telles par le composant C_i. Dans le cas de $Spotcheck_{i,j}$, les propriétés sont ajoutées à l'état des propriétés *pb* parce qu'elles sont crues par le composant C_i : elles n'ont été vérifiées que pour un échantillon X_{Ck}. La seule garantie procurée à C_i par $Spotcheck_{i,j}$ est qu'il a toujours la possibilité de détecter une erreur ou une fraude (mais il lui faut pour ce faire choisir un bon index, c'est-à-dire un index qui révélera l'erreur ou la fraude).

Les fonctions $\overline{Verif}^{Proof}$ et $\overline{Verif}^{Attest}$ définissent la sémantique des opérations de vérification correspondantes. Comme précisé précédemment, nous n'entrons pas dans les détails internes des vérifications des preuves et des attestations ici et supposons que $\overline{Verif}^{Proof}$ n'accepte que des preuves correctes et que $\overline{Verif}^{Attest}$ n'accepte que des attestations envoyées par un expéditeur authentique. La spécificité des événements $Verif_i^{Attest}$ est qu'ils ne génèrent de nouvelle connaissance que si l'auteur de l'attestation est considéré comme de confiance par le composant C_i (d'où la condition $Trust_{i,j} \in A$).

Nous notons aussi que les événements $Receive_{i,j}$ n'ajoutent pas de nouvelle connaissance en eux-mêmes car les propriétés reçues doivent être vérifiées avant de pouvoir être ajoutées aux états des propriétés.

Nous pouvons maintenant définir la sémantique d'une architecture A comme l'ensemble des états possibles produits par les traces compatibles avec cette architecture.

Définition 2 (Sémantique des architectures)**.** *La sémantique de l'architecture A est définie comme :* $\mathcal{S}(A) = \{\sigma \in State_{\perp}^{n} \mid \exists \theta \in T(A), S_T(\theta, Init^A) = \sigma\}$.

Dans ce qui suit, nous utilisons $\mathcal{S}_i(A)$ pour désigner le sous-ensemble de $\mathcal{S}(A)$ ne contenant que des états qui sont bien définis pour le composant C_i : $\mathcal{S}_i(A) = \{\sigma \in \mathcal{S}(A) \mid \sigma_i \neq Error\}$. L'ordre préfixe sur les traces entraîne l'ordre suivant sur les états : $\forall \sigma \in \mathcal{S}_i(A), \forall \sigma' \in \mathcal{S}_i(A), \sigma \geq_i \sigma' \Leftrightarrow \exists \theta \in T(A), \exists \theta' \in T(A), \sigma = S_T(\theta, Init^A), \sigma' = S_T(\theta', Init^A),$ et θ' est un préfixe de θ.

4.4 Propriétés

Les traces d'événements compatibles présentées précédemment servent à vérifier la satisfaction de certaines propriétés par les architectures correspondantes.

Le langage des propriétés et sa sémantique sont définis en 4.4.1. Une axiomatique permettant de prouver des propriétés d'architectures est présentée en 4.4.2 et ses preuves de correction et de complétude seront résumées en 4.4.3.

4.4.1 Langage des propriétés

Nous présentons dans le Tableau 4.6 la logique $\mathcal{L}_{FPP}$ [9] permettant de décrire les propriétés d'une architecture.

$$
\begin{array}{lll}
\phi ::= Has_i^{all}\left(\tilde{X}\right) & \mid Has_i^{none}\left(\tilde{X}\right) & \mid Has_i^{one}\left(\tilde{X}\right) \\
\quad \mid K_i\left(Eq\right) & \mid B_i\left(Eq\right) & \mid \phi_1 \wedge \phi_2 \\
Eq ::= T_1\ Rel\ T_2 \mid Eq_1 \wedge Eq_2 &&
\end{array}
$$

Tableau 4.6 – $\mathcal{L}_{FPP}$: langage des propriétés architecturales.

Le langage $\mathcal{L}_{FPP}$ implique deux modalités, notées K_i et B_i, qui représentent respectivement les propriétés de connaissance et de croyance d'un composant C_i. Elles font écho aux exigences d'intégrité rencontrées dans la démarche systématique (voir Section 3.4.3). Notons que la notation Eq (déjà utilisée dans le langage architectural) est utilisée ici pour représenter des conjonctions (plutôt que des ensembles) de relations primitives dans la logique.

La sémantique de $\mathcal{L}_{FPP}$ repose sur une approche dite de la « connaissance algorithmique déductive » dans laquelle la connaissance explicite d'un composant C_i est définie comme la connaissance que ce composant peut réellement calculer en utilisant un système déductif qui lui est associé [Puc04]. Cette relation déductive, $\triangleright_i$, est définie comme une relation entre un ensemble de propriétés Eq et une propriété Eq, soit $\left\{Eq^1, \ldots, Eq^n\right\} \triangleright_i Eq_0$. La relation $\triangleright_i$ peut typiquement être utilisée pour modéliser les propriétés des fonctions de la spécification. Par exemple, une propriété d'injectivité d'une fonction de hachage H s'exprime par $\{h_1 = H(x_1), h_2 = H(x_2), h_1 = h_2\} \triangleright_i (x_1 = x_2)$. Nous supposons que cette relation permet au moins à un composant de déduire chacune des propriétés d'un ensemble de propriétés ($\forall Eq \in E. E \triangleright_i Eq$) ainsi que toute conjonction $Eq_1 \wedge \cdots \wedge Eq_n$ de propriétés de cet ensemble. De plus, nous posons une hypothèse de transitivité de $\triangleright_i$: si $E \triangleright_i Eq^1, \ldots, E \triangleright_i Eq^n$ et $\left\{Eq^1, \ldots, Eq^n\right\} \triangleright_i Eq$ alors $E \triangleright_i Eq$.

$\mathcal{L}_{FPP}$ permet également d'exprimer d'autres propriétés utiles sur les architectures : par exemple $Has_i^{all}(\tilde{X})$ exprime le fait que le composant C_i peut obtenir ou dériver (en utilisant la relation de dépendance Dep_i) la valeur de X_k pour tous les k dans $Range(X)$. $Has_i^{one}(\tilde{X})$ exprime le fait que le composant C_i peut obtenir ou dériver la valeur de X_k pour un k dans $Range(X)$. Nous supposons $Range\ (X)$ suffisamment grand pour que $Has_i^{one}(\tilde{X})$ n'implique pas $Has_i^{all}(\tilde{X})$. Enfin, $Has_i^{none}(\tilde{X})$ est la propriété établissant que le composant C_i ne connaît aucune des valeurs X_k. Ces trois propriétés correspondent aux exigences de confidentialité définies dans l'étape de spécification (voir Section 3.4.3).

Nous soulignons ici que les propriétés Has_i signifient seulement que le composant C_i peut recevoir ou dériver des valeurs pour les variables concernées : elles n'apportent aucune garantie sur la correction de ces valeurs. Ces exigences d'intégrité s'expriment en utilisant les propriétés $K_i(Eq)$ et $B_i(Eq)$. $K_i(Eq)$ signifie que le composant C_i doit pouvoir établir la véracité de Eq

9. *Language of Formal Privacy Properties* en anglais.

tandis que $B_i(Eq)$ exprime le fait que le composant C_i doit pouvoir tester Eq, et donc détecter une violation éventuelle (en utilisant son système déductif $\triangleright_i$).

Nous pouvons maintenant définir la sémantique d'une propriété ϕ.

Définition 3 (Sémantique des propriétés.)**.** *La sémantique $S(\phi)$ d'une propriété ϕ de $\mathcal{L}_{FPP}$ est définie dans le Tableau 4.7 comme l'ensemble des architectures satisfaisant ϕ.*

$$
\begin{aligned}
A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right) &\Leftrightarrow \exists \sigma \in \mathcal{S}(A).[(\sigma_i^v(\tilde{X}) \neq \bot) \wedge \\
&\qquad (\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \Rightarrow \\
&\qquad\quad \forall l \in [1,k].(v_l \neq \bot)] \\
A \in S\left(Has_i^{none}\left(\tilde{X}\right)\right) &\Leftrightarrow \forall \sigma \in \mathcal{S}(A).(\sigma_i^v(\tilde{X}) = \bot) \\
A \in S\left(Has_i^{one}\left(\tilde{X}\right)\right) &\Leftrightarrow \forall \sigma \in \mathcal{S}(A).\{(\sigma_i^v(\tilde{X}) = \bot) \vee \\
&\qquad [\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \\
&\qquad\quad \nexists l, l' \in [1,k].(v_l \neq \bot) \wedge \\
&\qquad\quad (v_{l'} \neq \bot) \wedge (l \neq l')]\} \wedge \\
&\qquad \exists \sigma \in \mathcal{S}(A).[(\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \\
&\qquad\quad \exists l \in [1,k].(v_l \neq \bot)] \\
A \in S\left(K_i\left(Eq\right)\right) &\Leftrightarrow \forall \sigma' \in \mathcal{S}_i(A). \exists \sigma \in \mathcal{S}_i(A).(\sigma \geq_i \sigma') \wedge (\sigma_i^{pk} \triangleright_i Eq) \\
A \in S\left(B_i\left(Eq\right)\right) &\Leftrightarrow \forall \sigma' \in \mathcal{S}_i(A). \exists \sigma \in \mathcal{S}_i(A).(\sigma \geq_i \sigma') \wedge (\sigma_i^{pk} \cup \sigma_i^{pb} \triangleright_i Eq) \\
A \in S\left(\phi_1 \wedge \phi_2\right) &\Leftrightarrow A \in S(\phi_1) \wedge A \in S(\phi_2)
\end{aligned}
$$

Tableau 4.7 – Sémantique des propriétés de $\mathcal{L}_{FPP}$.

Une architecture satisfait la propriété $Has_i^{all}(\tilde{X})$ si et seulement si le composant C_i peut obtenir toutes les valeurs de $\tilde{X}$ dans au moins une trace d'exécution compatible tandis que $Has_i^{none}(\tilde{X})$ est satisfaite si et seulement si aucune trace d'exécution compatible ne peut aboutir à un état dans lequel le composant C_i peut associer une valeur à $\tilde{X}$ (ou à une partie de $\tilde{X}$ si $\tilde{X}$ est une variable tableau). $Has_i^{one}(\tilde{X})$ est quant à elle satisfaite si et seulement si aucune trace d'exécution compatible ne peut aboutir à un état dans lequel le composant C_i connaît plus d'une valeur du tableau $\tilde{X}$ et si le composant C_i peut obtenir au moins une valeur de $\tilde{X}$ dans une trace d'exécution compatible [10]. La validité des propriétés $K_i(Eq)$ et $B_i(Eq)$ est définie par référence aux traces d'exécution correctes (par rapport au composant C_i) puisqu'une trace incorrecte aboutit à un état dans lequel une erreur a été détectée par le composant. La condition $\sigma \geq \sigma'$ est utilisée pour écarter les états correspondant à des traces incomplètes dans lesquelles les propriétés Eq n'ont pas encore été établies. Comme expliqué précédemment, la capacité d'un composant C_i à dériver de nouvelles connaissances ou croyances est définie par son système déductif $\triangleright_i$.

10. Ces deux conditions cumulées font que le composant C_i ne peut obtenir qu'une seule valeur du tableau $\tilde{X}$.

4.4.2 Axiomatique des propriétés

Pour raisonner sur l'architecture et sur les connaissances et croyances des composants, nous introduisons dans le Tableau 4.8 une axiomatique de la logique présentée dans la section précédente. La satisfaction de la propriété ϕ par l'architecture A est notée $A \vdash \phi$.

$$\textbf{H1}\frac{Has_i\left(\tilde{X}\right) \in A}{A \vdash Has_i^{all}\left(\tilde{X}\right)} \qquad \textbf{H2}\frac{Receive_{i,j}\left(S,E\right) \in A \qquad \tilde{X} \in E}{A \vdash Has_i^{all}\left(\tilde{X}\right)}$$

$$\textbf{H3}\frac{Compute_G\left(\tilde{X}=T\right) \in A \qquad i \in G}{A \vdash Has_i^{all}\left(\tilde{X}\right)} \qquad \textbf{H4}\frac{Spotcheck_{i,j}\left(\tilde{X},E\right) \in A}{A \vdash Has_i^{one}\left(\tilde{X}\right)}$$

$$\textbf{H5}\frac{Dep_i\left(\tilde{Y},\tilde{X}\right) \qquad A \vdash Has_i^{all}\left(\tilde{X}\right)}{A \vdash Has_i^{all}\left(\tilde{Y}\right)} \qquad \textbf{H6}\frac{Dep_i\left(\tilde{Y},\tilde{X}\right) \qquad A \vdash Has_i^{one}\left(\tilde{X}\right)}{A \vdash Has_i^{one}\left(\tilde{Y}\right)}$$

$$\textbf{H7}\frac{A \nvdash Has_i^{all}\left(\tilde{X}\right) \qquad A \nvdash Has_i^{one}\left(\tilde{X}\right)}{A \vdash Has_i^{none}\left(\tilde{X}\right)}$$

$$\textbf{K1}\frac{Compute_G\left(\tilde{X}=T\right) \in A \qquad i \in G}{A \vdash K_i(\tilde{X}=T)} \qquad \textbf{K2}\frac{Check_i\left(E\right) \in A \qquad Eq \in E}{A \vdash K_i(Eq)}$$

$$\textbf{K3}\frac{Verif_i^{Proof}\left(Proof_j(E)\right) \in A \qquad Eq \in E}{A \vdash K_i(Eq)}$$

$$\textbf{K4}\frac{Verif_i^{Proof}\left(Proof_j(E)\right) \in A \qquad Attest_k(E') \in E \qquad Trust_{i,k} \in A \qquad Eq \in E'}{A \vdash K_i(Eq)}$$

$$\textbf{K5}\frac{Verif_i^{Attest}\left(Attest_j(E)\right) \in A \qquad Trust_{i,j} \in A \qquad Eq \in E}{A \vdash K_i(Eq)} \qquad \textbf{KB}\frac{A \vdash K_i(Eq)}{A \vdash B_i(Eq)}$$

$$\textbf{I}\wedge\ \frac{A \vdash \phi_1 \qquad A \vdash \phi_2}{A \vdash \phi_1 \wedge \phi_2} \qquad \textbf{B}\frac{Spotcheck_{i,j}\left(\tilde{X},E\right) \in A \qquad Eq \in E}{A \vdash B_i(Eq)}$$

$$\textbf{K}\triangleright\ \frac{E \triangleright_i Eq_0 \qquad \forall Eq \in E. A \vdash K_i(Eq)}{A \vdash K_i(Eq_0)} \qquad \textbf{B}\triangleright\ \frac{E \triangleright_i Eq_0 \qquad \forall Eq \in E. A \vdash B_i(Eq)}{A \vdash B_i(Eq_0)}$$

$$\textbf{K}\wedge\ \frac{A \vdash K_i(Eq_1) \qquad A \vdash K_i(Eq_2)}{A \vdash K_i(Eq_1 \wedge Eq_2)} \qquad \textbf{B}\wedge\ \frac{A \vdash B_i(Eq_1) \qquad A \vdash B_i(Eq_2)}{A \vdash B_i(Eq_1 \wedge Eq_2)}$$

Tableau 4.8 – Axiomatique des propriétés de $\mathcal{L}_{FPP}$.

Les règles (H1), (H2) et (H3) expriment que toutes les valeurs de $\tilde{X}$ peuvent être obtenues lorsque les primitives Has_i, $Receive_{i,j}$ et $Compute_G$ avec $i \in G$ sont présentes dans l'architecture. (H4) exprime le fait qu'une seule valeur au plus dans le tableau $\tilde{X}$ peut être obtenue via $Spotcheck_{i,j}$ dans l'architecture. (H5) et (H6) permettent de déduire les valeurs qui peuvent être obtenues par le composant C_i à partir des variables qu'il a déjà obtenues et de la relation de dépendance

Dep_i. (H7) modélise les cas où aucune des valeurs de $\tilde{X}$ ne peut être obtenue.

Les règles (K1), (K2), (K3), (K4) et (K5) permettent de déduire les connaissances qu'un composant peut obtenir à partir des primitives $Compute_G$, $Check_i$, $Proof_i$, $Attest_i$ et de leurs conditions de satisfaction respectives. La règle ($K\triangleright_i$) traduit les capacités de déduction de relations Eq d'un composant C_i à partir des relations qui lui sont déjà connues.

La règle (B) est utile pour dériver les croyances d'un composant à partir des $Spotcheck_{i,j}$ qu'il effectue. ($B\triangleright_i$) est similaire à ($K\triangleright_i$) mais s'applique aux relations crues par le composant C_i. (KB) indique que la modalité de connaissance K est plus forte que la modalité de croyance B.

Enfin, (K$\wedge$), (B$\wedge$) et (I$\wedge$) sont des règles structurelles permettant d'introduire l'opérateur conjonctif.

Exemple. Nous montrons dans les Tableaux 4.9 et 4.10 comment exprimer en $\mathcal{L}_{FPP}$ les exigences définies dans le chapitre précédent (voir Tableaux 3.9 et 3.10) pour les composants correspondants.

Variables	Has^{all}	Has^{none}
$ECONS_t$	C_U	C_P
$cons_t$	C_U	C_P
$price_t$	C_U	C_P
fee	C_U, C_P	

TABLEAU 4.9 – Spécification dans le langage $\mathcal{L}_{FPP}$ des exigences de confidentialité pour Ω_1.

Relations	K
$fee = \sum_t (price_t)$	C_U, C_P
$price_t = F(cons_t)$	C_U, C_P
$cons_t = S(ECONS_t)$	C_U, C_P

TABLEAU 4.10 – Spécification dans le langage $\mathcal{L}_{FPP}$ des exigences d'intégrité pour Ω_1.

Nous pouvons maintenant vérifier chacune des propriétés à l'aide de l'axiomatique. Les Tableaux 4.11 et 4.12 indiquent les règles permettant d'établir la validité des propriétés. Le premier élément de chaque paire représente l'entité concernée par la propriété et le second élément l'ensemble des règles la justifiant.

Variables	Has^{all}	Has^{none}
$ECONS_t$	(C_U, [H2, H5])	(C_P, [H7])
$cons_t$	(C_U, [H2])	(C_P, [H7])
$price_t$	(C_U, [H3])	(C_P, [H7])
fee	(C_U, [H3]), (C_P, [H2])	

TABLEAU 4.11 – Vérification des exigences de confidentialité pour Ω_1.

La vérification des propriétés de confidentialité Has^{all} s'appuie sur les règles correspondant aux opérations de communication (H2), de calcul (H3) et de dépendance à travers la relation *Dep* (H5). La vérification des propriétés de confidentialité Has^{none} s'appuie uniquement sur la règle (H7). Cette règle repose sur l'absence de primitives permettant d'obtenir un quelconque accès aux variables considérées.

Relations	K
$fee = \sum_t (price_t)$	(C_U, [K1]), (C_P, [K5])
$price_t = F(cons_t)$	(C_U, [K1]), (C_P, [K5])
$cons_t = S(ECONS_t)$	(C_U, [K5]), (C_P, [K5])

Tableau 4.12 – Vérification des exigences d'intégrité pour Ω_1.

La vérification des propriétés d'intégrité K repose uniquement sur les règles (K1) pour la confiance par le calcul et (K5) pour la confiance par attestation. Les types de confiance sélectionnés permettent ainsi aux exigences d'intégrité d'être satisfaites.

Cette dernière étape permet au concepteur de s'assurer que les exigences définies sont bien satisfaites. Si certaines ne l'étaient pas, il lui faudrait alors revenir dans les étapes de spécification ou de conception (voir Sections 3.4 et 3.5 respectivement).

4.4.3 Correction, complétude et décidabilité de l'axiomatique

Pour que l'axiomatique introduite dans le Tableau 4.8 soit effectivement utilisable en pratique, elle doit être correcte par rapport à la sémantique du Tableau 4.7. Dans cette section, nous donnons les preuves de correction, de complétude et de décidabilité de l'axiomatique.

Avant d'énoncer ces trois propriétés, nous établissons la définition de la couverture d'une trace et montrons que toute architecture cohérente admet au moins une trace couvrante (et cohérente). Nous montrerons ensuite que chaque mesure, calcul et réception d'une variable $\tilde{X}$ dans une trace (cohérente) compatible avec une architecture cohérente définit complètement la variable $\tilde{X}$ dans l'état $\sigma_i^v\left(\tilde{X}\right)$ du (ou des) composant(s) concerné(s).

Définition 4 (Couverture)**.** *Une trace θ de longueur $\overline{\theta}$ couvre une architecture A (notée $Couv^A(\theta)$) si et seulement si θ est compatible avec A et $\forall \alpha \in A$, $\exists a \in [1, \overline{\theta}], C(\theta_a, \alpha)$ avec C la fonction définie de correspondance (voir Définition 1).*

Intuitivement, une trace est couvrante si elle contient un événement correspondant à chaque primitive spécifiée dans l'architecture (au sens de la relation de correspondance C). Comme précédemment, nous ne nous intéressons toujours qu'à des architectures et des traces cohérentes.

Lemme 1 (Existence d'une trace couvrante)**.** *Toute architecture A (cohérente) possède au moins une trace θ couvrante (et cohérente).*

Démonstration. Toute architecture cohérente A est munie d'un ordre partiel $<<^A$ (voir Section 4.2.3 sur la cohérence des architectures).

Nous pouvons donc construire un ordre total arbitraire sur A par extension linéaire à partir de $<<^A$. L'architecture A étant finie, elle a donc un plus petit et un plus grand élément.

Cet ordre total permet de construire une trace θ compatible en transformant chaque élément α de A par un événement ϵ correspondant selon la relation de correspondance C. Pour ce faire, nous commençons par le plus petit élément de l'architecture selon l'ordre total issu de $<<^A$ et nous progressons vers le plus grand élément en ajoutant l'instance correspondant à chaque primitive à la fin de θ. Le fait que la relation d'ordre $<<^\theta$ soit cohérente avec $<<^A$ assure qu'il est toujours possible de dériver une trace compatible de cette manière (les valeurs introduites dans les événements de communication et de calcul pouvant être choisies de manière à refléter les événements précédents dans la trace).

Nous obtenons donc une trace θ couvrante (et cohérente puisque la relation d'ordre $<<^\theta$ est compatible avec $<<^A$ pour tous les éléments de θ et A correspondants). □

Nous montrons maintenant que la mesure, le calcul ou la réception d'une variable $\tilde{X}$ dans une trace (cohérente) compatible avec une architecture (cohérente) définit complètement la variable $\tilde{X}$ dans l'état $\sigma_i^v\left(\tilde{X}\right)$ du (ou des) composant(s) concerné(s).

Lemme 2 (Définition complète de l'état associé à des traces cohérentes pour les opérations de mesure, de calcul ou de réception.)**.** *Si θ est une trace cohérente et compatible avec A (aussi cohérente), alors $\forall a \in [1, \overline{\theta}]$ tel que*
$\left(\theta_a = Has_i\left(\tilde{X} : V\right) \vee \left(\theta_a = Compute_G\left(\tilde{X} = T\right) \wedge i \in G\right) \vee \theta_a = Receive_{i,j}\left(S, \{\tilde{X} : V\}\right)\right)$ *avec* $\sigma = S_T(\theta, Init^A)$ *implique* $\sigma_i^v\left(\tilde{X}\right) = V$ *est complètement défini.*

Démonstration. Nous raisonnons par récurrence généralisée sur la trace θ.

Cas de base. Soit $\dot{\theta}$ une trace vide. Le lemme est trivialement vrai pour $\dot{\theta}$ et on a donc $\sigma = S_T(\dot{\theta}, Init^A)$ implique $\sigma_i^v\left(\tilde{X}\right) = V$ est complètement défini.

Récurrence. Supposons une trace cohérente $\theta = \theta_1.\ldots.\theta_n$ qui satisfait le Lemme 2, c'est à dire telle que $\forall a \in [1, \overline{\theta}]$ on ait
$\theta_a = Has_i\left(\tilde{X} : V\right) \vee \left(\theta_a = Compute_G\left(\tilde{X} = T\right) \wedge i \in G\right) \vee \theta_a = Receive_{i,j}\left(S, \{\tilde{X} : V\}\right)$ avec $\sigma = S_T(\theta, Init^A)$ implique $\sigma_i^v\left(\tilde{X}\right) = V$ est complètement défini. Soit la trace $\theta' = \theta_1.\ldots.\theta_n.\theta_{n+1}$.

Si $\theta_{n+1} = Has_i\left(\tilde{X} : V\right)$, alors $\sigma' = S_T(\theta', Init^A)$ implique $\sigma_i'^v\left(\tilde{X}\right) = V$ est complètement défini par l'hypothèse selon laquelle les mesures sont bien définies.

Si $\theta_{n+1} = Receive_{i,j}\left(\{S\}, \{\tilde{X} : V\}\right)$, alors l'hypothèse de cohérence $HC5'$ impose que $\tilde{X}$ ait déjà été mesurée, reçue, ou calculée dans un des événements antérieurs et donc que $\sigma_j^v\left(\tilde{X}\right) = V$ est complètement défini. Par conséquent $\sigma' = S_T(\theta', Init^A)$ implique $\sigma_i'^v\left(\tilde{X}\right) = V \neq \perp$ par bonne définition de la réception.

Si $\theta_{n+1} = Compute_G\left(\tilde{X} = T\right)$ avec $i \in G$, alors l'hypothèse de cohérence $HC3'$ impose que toutes les variables apparaissant dans T aient déjà été mesurées, reçues, ou calculées dans un des événements antérieurs. Par conséquent, $\tilde{X} = \epsilon\left(T, \bigcup_{i \in G} \sigma_i^v\right)$ peut être calculé et nous avons

$\sigma_i'^{v}\left(\tilde{X}\right) = V$ pour les $i \in G$ est complètement défini avec $\sigma' = S_T(\theta', Init^A)$ par bonne définition des calculs.

Si θ_{n+1} correspond à un autre événement, comme la définition des σ_i^v est monotone (aucun événement ne peut assigner $\perp$ à un $\sigma_i^v\left(\tilde{X}\right)$), alors $\sigma' = S_T(\theta', Init^A)$ implique $\sigma_i'^{v}\left(\tilde{X}\right) = V$ est complètement défini.

Nous avons donc bien $\forall a \in [1, \overline{\theta'}]$ tel que
$\left(\theta_a = Has_i\left(\tilde{X} : V\right) \vee \left(\theta_a = Compute_G\left(\tilde{X} = T\right) \wedge i \in G\right) \vee \theta_a = Receive_{i,j}\left(S, \{\tilde{X} : V\}\right)\right)$ avec $\sigma' = S_T(\theta', Init^A)$ implique $\sigma_i'^{v}\left(\tilde{X}\right) = V$ est complètement défini, ce qui prouve la propriété de la définition complète d'une trace pour les valeurs mesurées, calculées ou reçues. □

Nous pouvons désormais établir la propriété de correction.

Proposition 1 (Correction)**.** *Quelle que soit l'architecture A de Γ, si $A \vdash \phi$ alors $A \in S(\phi)$.*

La propriété de correction est prouvée par récurrence sur les arbres de preuve de $A \vdash \phi$. Nous considérons tour à tour chaque règle du Tableau 4.8 et montrons que, si leurs prémisses satisfont la propriété de correction, leur conclusion $A \vdash \phi$ est telle que $A \in S(\phi)$.

Par abus de notation nous utilisons la notation $\epsilon \in \theta$ pour représenter la propriété $\exists a \in \left[1, \overline{\theta}\right].\theta_a = \epsilon$. Nous pouvons maintenant établir les preuves de correction pour chacune des règles de l'axiomatique.

Démonstration de la propriété de correction. **Cas de H1 :** Si $Has_i\left(\tilde{X}\right) \in A$, alors $\exists \theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. On a donc $\exists V \in Val.Has_i\left(\tilde{X} : V\right) \in \theta$ selon la définition d'une trace couvrante (Définition 4). Par conséquent, nous avons $\exists \sigma \in State_\perp^n.\exists \theta \in T(A).\sigma = S_T\left(\theta, Init^A\right)$ tel que $\exists V \in Val_\perp.\sigma_i^v\left(\tilde{X}\right) = V$ en vertu de la sémantique des traces d'événements (Tableau 4.5). Or, comme $Has_i\left(\tilde{X} : V\}\right) \in \theta$ alors $V \in Val$ selon le Lemme 2. et $\sigma_i^v\left(\tilde{X}\right)$ est complètement défini. Nous avons alors $\exists \sigma \in \mathcal{S}(A).\sigma_i^v\left(\tilde{X}\right)$ est complètement défini par la sémantique des architectures (Définition 2). Enfin, nous avons bien $A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de H2 : Si $Receive_{i,j}(S, E) \in A$, alors $\exists \theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. On a donc $\exists V \in Val_\perp.Receive_{i,j}\left(S, \{\tilde{X} : V\}\right) \in \theta$ selon la définition d'une trace couvrante (Définition 4). Par conséquent, nous avons $\exists \sigma \in State_\perp^n.\exists \theta \in T(A).\sigma = S_T\left(\theta, Init^A\right)$ tel que $\exists V \in Val_\perp.\{\sigma_i^v\left(\tilde{X}\right) = V\}$ en vertu de la sémantique des traces d'événements (Tableau 4.5). Or, comme $Receive_{i,j}\left(S, \{\tilde{X} : V\}\right) \in \theta$ alors $V \in Val$ selon le Lemme 2 et $\sigma_i^v\left(\tilde{X}\right)$ est complètement défini. Nous en déduisons donc que $A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de H3 : Si $Compute_G\left(\tilde{X} = T\right) \in A$ avec $i \in G$, alors $\exists \theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. On a donc $Compute_G\left(\tilde{X} = T\right) \in \theta$ selon la définition d'une trace couvrante (Définition 4). Par conséquent, nous avons $\exists \sigma \in State_\perp^n.\exists \theta \in T(A).\sigma = S_T\left(\theta, Init^A\right)$ tel que $\sigma_i^v\left(\tilde{X}\right) = \varepsilon\left(T, \sigma_i^v\right)$ en vertu de la sémantique des traces d'événements (Tableau 4.5). Or,

comme $Compute_G\left(\tilde{X} = T\right) \in \theta$ avec $i \in G$ alors $\sigma_i^v\left(\tilde{X}\right)$ est complètement défini selon le Lemme 2. Nous en déduisons donc que $A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de H4 : Si $Spotcheck_{i,j}\left(\tilde{X}, Eq\right) \in A$, alors $\exists\theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. On a donc $\exists V \in Val.Spotcheck_{i,j}(X_{Ck} : V, Eq) \in \theta$ selon la définition d'une trace couvrante (Définition 4). Par conséquent, nous avons $\exists\sigma \in State_\perp^n.\exists\theta \in T(A).\sigma = S_T\left(\theta, Init^A\right)$ tel que $\exists V \in Val.\sigma_i^v(X_{Ck}) = V$ en vertu de la sémantique des traces d'événements (Tableau 4.5). Or la définition de la compatibilité (Définition 1) interdit d'avoir tout autre *Spotcheck* entre C_i et C_j et il n'est pas possible d'obtenir la valeur d'un élément quelconque de X d'une autre manière pour C_i par cohérence par (HC10). Nous avons alors $\forall\sigma \in \mathcal{S}(A).\{\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k >$ avec $\nexists l, l' \in [1,k].[(v_l \neq \perp) \wedge (v_{l'} \neq \perp) \wedge (l \neq l')]\}$ et $\exists\sigma \in \mathcal{S}(A).(\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k >$ avec $\exists l \in [1,k].(v_l \neq \perp))$ par la sémantique des architectures (Définition 2). Enfin, nous avons bien $A \in S\left(Has_i^{one}\left(\tilde{X}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de H5 : Si $Dep_i\left(\tilde{Y}, \tilde{X}\right)$, alors $\exists \mathrm{F} \in Fun.\mathrm{F}\left(\tilde{X}\right) = \tilde{Y}$ avec $\tilde{X}$ complètement défini et F totale par hypothèse de cohérence de la relation de dépendance (HCDep). Nous savons que $\exists\theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. Il y a donc deux possibilités : soit $Compute_G\left(\tilde{Y} = T\right) \in A$ avec $i \in G$ et on a $Compute_G\left(\tilde{Y} = T\right) \in \theta'$ avec $\theta' = \theta$, soit $Compute_G\left(\tilde{Y} = T\right) \notin A$ et on peut construire de manière cohérente $\theta' = \theta.Compute_i\left(\tilde{Y} = T\right)$ avec $T = \mathrm{F}\left(\tilde{X}\right)$ car $\tilde{X}$ sont bien définies $(A \vdash Has_i^{all}\left(\tilde{X}\right))$ par hypothèse pour cette règle. Dans tous les cas, on a donc $\exists\theta' \in T(A).\left(Couv^A(\theta') \wedge Compute_G\left(\tilde{Y} = T\right) \in \theta'\right)$ avec $i \in G$ selon la définition d'une trace compatible (Définition 1). Nous pouvons donc en déduire que $A \in S\left(Has_i^{all}\left(\tilde{Y}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7) de la même manière que pour le cas de H3.

Cas de H6 : Si $Dep_i\left(\tilde{Y}, \tilde{X}\right)$, alors $\exists \mathrm{F} \in Fun.\mathrm{F}\left(\tilde{X}\right) = \tilde{Y}$ avec $\tilde{X}$ complètement défini et F totale par hypothèse de cohérence de la relation de dépendance (HCDep). Nous savons que $\exists\theta \in T(A).Couv^A(\theta)$ d'après le Lemme 1. Comme les calculs ne peuvent être exprimés que sur des tableaux entiers, il n'y a qu'une seule possibilité : on peut construire de manière cohérente $\theta' = \theta.Compute_i(Y_{Ck} = T)$ avec $T = \mathrm{F}(X_{Ck})$ et X_{Ck} défini $(A \vdash Has_i^{one}\left(\tilde{X}\right)$ par hypothèse pour cette règle) selon la définition d'une trace compatible (Définition 1). On a donc $\exists\sigma \in \mathcal{S}(A).(\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k >$ avec $\exists l \in [1,k].(v_l \neq \perp))$ pour $\sigma = S_T\left(\theta', Init^A\right)$. De plus, la définition de la compatibilité (Définition 1) interdit d'avoir tout autre *Spotcheck* entre C_i et C_j et il n'est pas possible d'obtenir la valeur d'un élément quelconque de $\tilde{X}$ d'une autre manière pour C_i par cohérence par (HC10). Nous avons alors $\forall\sigma \in \mathcal{S}(A).\{\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k >$ avec $\nexists l, l' \in [1,k].[(v_l \neq \perp) \wedge (v_{l'} \neq \perp) \wedge (l \neq l')]\}$ par la sémantique des architectures (Définition 2). Enfin, nous avons bien $A \in S\left(Has_i^{one}\left(\tilde{X}\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de H7 : Nous allons appliquer un raisonnement par contraposée : supposons que si $A \notin S\left(Has_i^{none}\left(\tilde{X}\right)\right)$ alors $A \nvdash Has_i^{none}\left(\tilde{X}\right)$. Nous avons alors $\exists\sigma \in \mathcal{S}(A).\sigma_i^v\left(\tilde{X}\right) \neq \perp$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7). Nous en déduisons donc que $\exists\sigma \in State_\perp^n.\exists\theta \in T(A).\sigma = S_T\left(\theta, Init^A\right)$ tel que $\sigma_i^v\left(\tilde{X}\right) \neq \perp$ par la sémantique des architectures (Définition 2). Or $A \nvdash Has_i^{all}\left(\tilde{X}\right)$ et $A \nvdash Has_i^{one}\left(\tilde{X}\right)$ alors que ces propriétés capturent

tous les cas qui permettent de changer la valeur initiale d'une variable (qui est $\perp$) à travers les primitives architecturales et les événements compatibles (Has_i, $Compute_G$ avec $i \in G$, $Receive_{i,j}$ et $Spotcheck_{i,j}$ selon le Tableau 4.5). Donc $\sigma_i^v\left(\tilde{X}\right) = \perp$. Nous parvenons donc à une contradiction qui nous permet de conclure que $A \in S\left(Has_i^{none}\left(\tilde{X}\right)\right)$.

Cas de K1 : Si $Compute_G\left(\tilde{X} = T\right) \in A$ avec $i \in G$, alors nous avons $\forall\theta' \in T(A).\exists\theta \in T(A).\left(Couv^A(\theta) \wedge \theta \geq \theta' \wedge Compute_G\left(\tilde{X} = T\right) \in \theta\right)$ car toute trace compatible peut être étendue en une trace couvrante. Par conséquent, nous en déduisons que $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\exists\theta' \in T(A).\exists\theta \in Couv(A)$ et $\left(Couv^A(\theta) \wedge \theta \geq \theta' \wedge \sigma' = S_T\left(\theta', Init^A\right) \wedge \sigma = S_T\left(\theta, Init^A\right) \wedge \left(\tilde{X} = T\right) \in \sigma_i^{pk}\right)$ d'après la sémantique des traces (Tableau 4.5) et la sémantique des architectures (Définition 2) car les calculs se font sans erreur par hypothèse. Nous avons donc $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge \left(\tilde{X} = T\right) \in \sigma_i^{pk}\right)$ par correspondance entre l'ordre préfixe des traces et l'ordre sur les états. Par conséquent, $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A)$ tel que $\left(\sigma \geq_i \sigma' \wedge \sigma_i^{pk} \triangleright_i \left(\tilde{X} = T\right)\right)$ puisque $\triangleright_i$ permet par hypothèse à un composant de déduire chacune des propriétés d'un ensemble de propriétés. Nous avons donc bien $A \in S\left(K_i\left(\tilde{X} = T\right)\right)$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de K2 : Comme pour *K1* en remplaçant $Compute_G\left(\tilde{X} = T\right)$ par $Check_i(E)$ et en considérant tous les Eq tel que $Eq \in E$. Si $Check_i(E) \in A$, alors nous avons $\forall\theta' \in T(A).\exists\theta \in T(A).\left(Couv^A(\theta) \wedge \theta \geq \theta' \wedge Check_i(E) \in \theta\right)$ car toute trace compatible peut être étendue en une trace couvrante. Par conséquent, nous en déduisons que $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\exists\theta' \in T(A).\exists\theta \in Couv(A)$ et $\left(Couv^A(\theta) \wedge \theta \geq \theta' \wedge \sigma' = S_T\left(\theta', Init^A\right) \wedge \sigma = S_T\left(\theta, Init^A\right) \wedge E \subseteq \sigma_i^{pk}\right)$ d'après la sémantique des traces (Tableau 4.5), la sémantique des architectures (Définition 2) et le fait que les valeurs cohérentes d'une trace garantissent que $\epsilon(Eq, \sigma_i^v) = True$ pour tous les $Eq \in E$ (aboutissant à l'absence d'erreur dans l'état). Nous avons donc $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge E \subseteq \sigma_i^{pk}\right)$ par correspondance entre l'ordre préfixe des traces et l'ordre sur les états. Par conséquent, $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge \sigma_i^{pk} \triangleright_i \mathrm{Eq}\right)$ pour tous $Eq \in E$ puisque $\triangleright_i$ permet par hypothèse à un composant de déduire chacune des propriétés d'un ensemble de propriétés. Nous avons donc bien $A \in S(K_i(Eq))$ pour tous les $Eq \in E$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de K3 : Comme pour le cas de (K2) en remplaçant $Check_i(E)$ par $Verif_i^{Proof}\left(Proof_j(E)\right)$.

Cas de K4 : Comme pour le cas de (K3) en considérant que $Attest_k(E') \in E$ avec $Eq \in E'$ et $Trust_{i,j} \in A$.

Cas de K5 : Comme pour le cas de (K2) en remplaçant $Check_i(E)$ par $Verif_i^{Attest}\left(Attest_j(E)\right)$ avec $Trust_{i,j} \in A$.

Cas de K$\wedge$: Si $A \vdash K_i(Eq_1)$ et $A \vdash K_i(Eq_2)$ alors $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A)$ tel que $\left(\sigma \geq_i \sigma' \wedge \sigma_i^{pk} \triangleright_i Eq_1 \wedge \sigma_i^{pk} \triangleright_i Eq_2\right)$ d'après la sémantique des traces (Tableau 4.5) et la sémantique des architectures (Définition 2). Or par hypothèse $\triangleright_i$ permet de déduire les conjonctions de ses déductions, on a donc $\sigma_i^{pk} \triangleright_i (Eq_1 \wedge Eq_2)$ et donc $A \in S(K_i(Eq_1 \wedge Eq_2))$.

Cas de K$\triangleright$: Si pour tous les $Eq_1, \ldots, Eq_n \in E$ on a $A \vdash K_i(Eq_l)$ pour $l \in [1, n]$ alors on a $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma_1 \in \mathcal{S}_i(A)$ tel que $(\sigma_1 \geq_i \sigma') \wedge \left(\sigma_{1i}^{pk} \triangleright_i Eq_1\right)$ d'après la sémantique des traces

(Tableau 4.5) et la sémantique des architectures (Définition 2). On peut alors établir que $\exists\sigma_2 \in \mathcal{S}_i(A)$ tel que $(\sigma_2 \geq_i \sigma_1) \wedge \left(\sigma_{2i}^{pk} \triangleright_i (Eq_1 \wedge Eq_2)\right)$ car l'ajout de propriétés aux états est une fonction monotone (il n'est pas possible pour un composant d'oublier une propriété). En procédant ainsi pour tous les Eq_l jusqu'à $l = n$, on obtient que $\forall Eq_1, \cdots \in E.\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma_n \in \mathcal{S}_i(A)$ tel que $\left((\sigma_n \geq_i \sigma') \wedge \left(\bigwedge_{l\in[1,n]} \sigma_{ni}^{pk} \triangleright_i Eq_l\right)\right)$ et on a donc $\sigma_{ni}^{pk} \triangleright_i \left(\bigwedge_{l\in[1,n]} Eq\right)$ par l'hypothèse de déduction d'une conjonction par $\triangleright_i$. Par l'hypothèse de transitivité de $\triangleright_i$ et l'hypothèse du cas $E \triangleright_i Eq_0$ nous pouvons déduire $\sigma_{ni}^{pk} \triangleright_i Eq_0$. Nous avons donc bien $A \in S\left(K_i\left(Eq_0\right)\right)$.

Cas de KB : Si $A \vdash K_i\left(Eq\right)$ alors $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge \sigma_i^{pk} \triangleright_i Eq\right)$ d'après la sémantique des traces (Tableau 4.5) et la sémantique des architectures (Définition 2). Or $\forall\sigma_i^{pb}.\sigma_i^{pk} \subseteq \left(\sigma_i^{pk} \cup \sigma_i^{pb}\right)$ donc si $\sigma_i^{pk} \triangleright_i Eq$ alors $\left(\sigma_i^{pk} \cup \sigma_i^{pb}\right) \triangleright_i Eq$. Nous pouvons donc déduire que $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge \left(\sigma_i^{pk} \cup \sigma_i^{pb}\right) \triangleright_i Eq\right)$. Par conséquent, nous avons bien $A \in S\left(B_i\left(Eq\right)\right)$.

Cas de B : Si $Spotcheck_{i,j}\left(\tilde{X}, E\right) \in A$, alors nous avons $\forall\theta' \in T(A).\exists\theta \in T(A).\left(Couv^A\left(\theta\right) \wedge \theta \geq \theta' \wedge \exists V.\exists Ck.Spotcheck_{i,j}\left(X_{Ck:V}, E\right) \in \theta\right)$ car toute trace compatible peut être étendue en une trace couvrante. Par conséquent, nous en déduisons que $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\exists\theta' \in T\left(A\right).\exists\theta \in Couv\left(A\right)$ et $\left(Couv^A\left(\theta\right) \wedge \theta \geq \theta' \wedge \sigma' = S_T\left(\theta', Init^A\right) \wedge \sigma = S_T\left(\theta, Init^A\right) \wedge E \subseteq \sigma_i^{pb}\right)$ d'après la sémantique des traces (Tableau 4.5), la sémantique des architectures (Définition 2) et l'hypothèse de cohérence (HC12') qui garantit que $\epsilon\left(Eq[\tilde{X}/X_{Ck}], \sigma_i^v[X_{Ck}/V]\right) = True$ pour tous les $Eq \in E$ (état sans erreur). Nous avons donc $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge E \subseteq \sigma_i^{pb}\right)$ par correspondance entre l'ordre préfixe des traces et l'ordre sur les états. Par conséquent, $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq_i \sigma' \wedge \sigma_i^{pb} \triangleright_i \mathrm{Eq}\right)$ pour tous $Eq \in E$ puisque $\triangleright_i$ permet par hypothèse à un composant de déduire chacune des propriétés d'un ensemble de propriétés. Nous avons donc bien $A \in S\left(B_i\left(Eq\right)\right)$ pour tous les $Eq \in E$ d'après la sémantique de $\mathcal{L}_{FPP}$ (Tableau 4.7).

Cas de B$\wedge$: Comme pour $K\wedge$ en remplaçant σ_i^{pk} par σ_i^{pb} après application de (KB) pour tous les éléments de σ_i^{pk}.

Cas de B$\triangleright$: Comme pour $K\triangleright$ en remplaçant σ_i^{pk} par σ_i^{pb} après application de (KB) pour tous les éléments de σ_i^{pk}.

Cas de I$\wedge$: Si $A \vdash \phi_1$ et $A \vdash \phi_2$ alors $A \in S(\phi_1)$ et $A \in S(\phi_2)$. On a alors directement $A \in S(\phi_1 \wedge \phi_2)$ par la Définition 3 (sémantique des propriétés).

□

Nous pouvons maintenant énoncer la propriété de complétude.

Proposition 2 (Complétude)**.** *Quelle que soit l'architecture A de* Γ*, si* $A \in S\left(\phi\right)$ *alors* $A \vdash \phi$*.*

La propriété de complétude est prouvée par une analyse de cas du Tableau 4.7 qui définit $A \in S(\phi)$, analyse qui fait elle-même appel à la définition de la sémantique des traces du Tableau 4.5.

Démonstration de la propriété de complétude. **Cas de Has_i^{all} :** Si $A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right)$, alors $\exists\sigma \in \mathcal{S}(A).\sigma_i^v(\tilde{X})$ est complètement défini d'après la sémantique des propriétés (Tableau 4.7). Par conséquent, nous avons $\exists\sigma \in \mathcal{S}(A).\exists\theta \in T(A).\left(\sigma = S_T(\theta, Init^A) \wedge \sigma_i^v\left(\tilde{X}\right) = V\right)$ est complètement défini d'après la sémantique d'une architecture (Définition 2). Mais $\sigma = S_T(\theta, Init^A)$ et $\sigma_i^v\left(\tilde{X}\right) = V$ est complètement défini implique que $\exists V \in Val.Has_i\left(\tilde{X} : V\right) \in \theta$ ou $Receive_{i,j}\left(S, \{\tilde{X} : V\}\right) \in \theta$ ou $Compute_G\left(\tilde{X} = T\right) \in \theta$ avec $i \in G$ par la définition de S_T (Tableau 4.5). Nous en déduisons que $Has_i\left(\tilde{X}\right) \in A$ ou bien $Receive_i\left(S, \{\tilde{X}\}\right) \in A$ ou bien $Compute_G\left(\tilde{X} = T\right) \in A$ avec $i \in G$ ou bien $Dep_i\left(\tilde{X}, \tilde{Y}\right)$ avec $Has_i^{all}\left(\tilde{Y}\right)$ par les définitions de trace compatible (Définition 1) et de cohérence. Dans chaque cas, on peut conclure que $A \vdash Has_i^{all}\left(\tilde{X}\right)$ en utilisant respectivement (H1), (H2), (H3) ou (H5).

Cas de Has_i^{none} : Si $A \in S\left(Has_i^{none}\left(\tilde{X}\right)\right)$, alors $\forall\sigma \in \mathcal{S}(A).\sigma_i^v(\tilde{X}) = \perp$ d'après la sémantique des propriétés (Tableau 4.7). Par conséquent, nous avons $\forall\sigma \in \mathcal{S}(A).\forall\theta \in T(A)$ tel que $\sigma = S_T(\theta, Init^A)$ implique $\sigma_i^v\left(\tilde{X}\right) = \perp$ d'après la sémantique d'une architecture (Définition 2). Comme $\sigma_i^v\left(\tilde{X}\right) = \perp$ initialement et que tout événement modifiant $\sigma_i^v\left(\tilde{X}\right)$ lui donnerait une valeur différente de $\perp$, aucun des événements modifiant $\sigma_i^v\left(\tilde{X}\right)$ ne doit apparaître dans la trace par la définition de S_T (Tableau 4.5). On a donc $Has_i\left(\tilde{X} : V\right) \notin \theta$ et $Receive_{i,j}\left(S, \{\tilde{X} : V\}\right) \notin \theta$ et $Compute_G\left(\tilde{X} = T\right) \notin \theta$ avec $i \in G$ et $Spotcheck_{i,j}\left(X_{Ck} : V, Eq\right) \notin \theta$. Nous en déduisons que $Has_i\left(\tilde{X}\right) \notin A$ et $Receive_i\left(S, \{\tilde{X}\}\right) \notin A$ et $Compute_G\left(\tilde{X} = T\right) \notin A$ avec $i \in G$ et $Spotcheck_{i,j}\left(\tilde{X}, E\right) \notin A$ et $Dep_i\left(\tilde{X}, \tilde{Y}\right)$ faux ou ni $A \vdash Has_i^{all}\left(\tilde{Y}\right)$ ni $A \vdash Has_i^{one}\left(\tilde{Y}\right)$ par les définitions de trace compatible (Définition 1) et de cohérence. Dans ce cas, on peut conclure que $A \vdash Has_i^{none}\left(\tilde{X}\right)$ en utilisant (H1), (H2), (H3), (H4), (H5), (H6) et (H7).

Cas de Has_i^{one} : Si $A \in S\left(Has_i^{one}\left(\tilde{X}\right)\right)$, alors $\forall\sigma \in \mathcal{S}(A).\sigma_i^v(\tilde{X}) = \perp \vee \left(\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \nexists\left(l, l'\right), l \neq l', v_l \neq \perp, v_{l'} \neq \perp\right)$ d'après la sémantique des propriétés (Tableau 4.7). Par conséquent, nous avons $\forall\sigma \in \mathcal{S}(A).\forall\theta \in T(A)$ avec $\sigma = S_T(\theta, Init^A)$ implique $\sigma_i^v\left(\tilde{X}\right) = \perp \vee (\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \nexists\left(l, l'\right), l \neq l', v_l \neq \perp, v_{l'} \neq \perp)$ d'après la sémantique d'une architecture (Définition 2). Comme $\sigma_i^v\left(\tilde{X}\right) = \perp$ initialement et que tout événement modifiant $\sigma_i^v\left(\tilde{X}\right)$ lui donnerait une valeur différente de $\perp$, aucun des événements définissant $\sigma_i^v\left(\tilde{X}\right)$ pour plus d'un élément ne doit apparaître dans la trace par la définition de S_T (Tableau 4.5). On a donc $Has_i\left(\tilde{X} : V\right) \notin \theta$ et $Receive_{i,j}\left(S, \{\tilde{X} : V\}\right) \notin \theta$ et $Compute_G\left(\tilde{X} = T\right) \notin \theta$ avec $i \in G$. Nous en déduisons que $Has_i\left(\tilde{X}\right) \notin A$ et $Receive_i\left(S, \{\tilde{X}\}\right) \notin A$ et $Compute_G\left(\tilde{X} = T\right) \notin A$ avec $i \in G$ et soit $Dep_i\left(\tilde{X}, \tilde{Y}\right)$ est faux soit $A \vdash Has_i^{all}\left(\tilde{Y}\right)$ est faux par les définitions de trace compatible (Définition 1) et de cohérence. Deux cas sont alors possibles. Soit $Spotcheck_{i,j}\left(\tilde{X}, E\right) \in A$ ou bien $Dep_i\left(\tilde{X}, \tilde{Y}\right\}$ et $Has_i^{one}\left(\tilde{Y}\right)$ et il est alors possible de déduire que $A \vdash Has_i^{one}\left(\tilde{X}\right)$ par (H4) ou bien par (H6). Soit $Spotcheck_{i,j}\left(\tilde{X}, E\right) \notin A$ et il n'est alors pas possible de

satisfaire la seconde partie de la conjonction principale dans la sémantique de Has^{one} : $\exists\sigma \in \mathcal{S}(A).[(\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \exists l \in [1,k].(v_l \neq \perp)]$.

Cas de K_i : Si $A \in S(K_i(Eq))$, alors $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\left(\sigma \geq \sigma' \wedge \sigma_i^{pk} \triangleright_i Eq\right)$ d'après la sémantique des propriétés (Tableau 4.7). Par conséquent, nous avons $\forall\sigma' \in \mathcal{S}_i(A).\exists\sigma \in \mathcal{S}_i(A).\exists\theta \in T(A).\sigma = S_T(\theta, Init^A) \wedge \left(\sigma \geq \sigma' \wedge \sigma_i^{pk} \triangleright_i Eq\right)$ d'après la sémantique d'une architecture (Définition 2). Mais $\sigma = S_T(\theta, Init^A) \wedge \sigma_i^{pk} \triangleright_i Eq$ implique soit que $Compute_G\left(\tilde{X} = T\right) \in \theta$ avec $\tilde{X} = T$ et $i \in G$ pour Eq, soit que $Check_i(E) \in \theta$ avec $Eq \in E$, soit que $Verif_i^{Proof}\left(Proof_j(E)\right) \in \theta$ avec $Eq \in E$, soit que $Verif_i^{Proof}\left(Proof_j(E)\right) \in \theta$ avec $Attest_k(E') \in E$, $Trust_{i,j'} \in \sigma_i^{pk}$ et $Eq \in E'$, soit que $Verif_i^{Attest}\left(Attest_j(E)\right) \in A$ et $Trust_{i,j'} \in \sigma_i^{pk}$ par la définition de S_T (Tableau 4.5) car ce sont les seuls événements modifiant σ_i^{pk}. Nous en déduisons soit que $Compute_G\left(\tilde{X}\right) \in A$ avec $\tilde{X} = T$ et $i \in G$ pour Eq, soit que $Check_i(E) \in A$ avec $Eq \in E$, soit que $Verif_i^{Proof}\left(Proof_j(E)\right) \in A$ avec $Eq \in E$, soit que $Verif_i^{Proof}\left(Proof_j(E)\right) \in A$ avec $Attest_k(E') \in E$, $Trust_{i,k} \in A$ et $Eq \in E'$, soit que $Verif_i^{Attest}\left(Attest_j(E)\right) \in A$ et $Trust_{i,j} \in A$, soit enfin que $E \triangleright_i Eq_0$ et $\forall Eq \in E.A \vdash K_i(Eq)$ par les définitions de trace compatible (Définition 1) et de cohérence. Soit Eq est une équation qui apparaît directement dans une des primitives architecturales et on peut conclure que $A \vdash K_i(Eq)$ en utilisant (K1), (K2), (K3), (K4) ou (K5), soit Eq appartient à l'ensemble des primitives dérivables par la relation de déduction $\triangleright_i$ et on peut alors déduire $A \vdash K_i(Eq)$ par application de (K$\wedge$) et (K$\triangleright$).

Cas de B_i : Comme pour le cas de K_i en considérant en plus que l'on peut aussi avoir $Spotcheck_{i,j}(X_{Ck} : V, Eq) \in \theta$ car tous les événements modifiant σ_i^{pk} et σ_i^{pb} doivent être considérés. Il est donc aussi possible d'avoir $Spotcheck_{i,j}\left(\tilde{X}, Eq\right) \in A$. On peut donc en déduire que $A \vdash B_i(Eq)$ en utilisant d'abord (K1), (K2), (K3), (K4), (K5) et (KB), (B), (B$\wedge$) et (B$\triangleright$).

Cas de $\wedge$: Si $A \in S(\phi_1 \wedge \phi_2)$ alors $A \in S(\phi_1) \wedge A \in S(\phi_2)$ d'après la sémantique des propriétés (Tableau 4.7). Pour toutes les formes (Has_i^{all}, Has_i^{all}, Has_i^{all}, K_i et B_i) de ϕ_1 et ϕ_2 on peut déduire $A \vdash \phi_1$ et $A \vdash \phi_1$. On dérive donc $A \vdash \phi_1 \wedge \phi_2$ par (I$\wedge$). On procède par récurrence structurelle sur le langage des propriétés architecturales $\mathcal{L}_{FPP}$ pour les propriétés ϕ_1 et ϕ_2 sous forme conjonctive. □

Remarque. Nous avons restreint les fonctions F s'appliquant aux termes à des fonctions unaires dans T et T^ϵ (voir Tableaux 4.1 et 4.3). Cette restriction pourrait être levée mais cela empêcherait d'obtenir la complétude de l'axiomatique (voir Tableau 4.8). En effet, dans le cas d'une égalité de la forme $X = F(Y, Z)$, la règle (H6) ne pourrait être appliquée car la connaissance d'une valeur de chacune des variables Y et Z pourrait, dans certains cas, permettre de connaître deux valeurs du tableau X (dans le cas où F admet un élément absorbant par exemple). Il faudrait alors imposer qu'une seule des variables ait un élément connu ($A \vdash Has_i^{one}$) et que toutes les autres variables n'en aient pas ($A \vdash Has_i^{none}$). On utiliserait alors ($A \vdash Has_i^{none}$) comme prémisse de la règle (H6). Or la règle (H7) qui définit les $A \vdash Has_i^{none}$ a pour prémisse $A \nvdash Has_i^{one}$, ce qui suppose qu'ils soient tous dérivables avant d'appliquer (H7). Un moyen de simuler une fonctions n-aires dans ce cadre est d'utiliser une fonction de pairage [11] telle qu'une fonction de Cantor.

11. *Pairing function* en anglais.

Enfin, nous énonçons la propriété de décidabilité comme suit.

Proposition 3 (Décidabilité)**.** *Si les systèmes déductifs $\triangleright_i$ sont décidables, alors l'axiomatique du Tableau 4.8 est également décidable.*

La propriété de décidabilité est prouvée sur la base de la stratification des preuves de Has_i^{all}, Has_i^{none}, Has_i^{one}, K_i et B_i. On considère d'abord les preuves de propriétés ne faisant pas intervenir les systèmes déductifs $\triangleright_i$ des composants, puis les applications des $\triangleright_i$. La preuve détaillée est donnée par l'existence d'un algorithme présenté dans le chapitre présentant l'implémentation (voir Section 5.3).

4.5 Conclusion

Le cadre formel présenté dans ce chapitre [ALM14a] répond aux objectifs fixés en introduction : définitions précises, fondements pour des raisonnements sûrs et documentation justifiant les choix de conception.

Cependant, les concepteurs ne sont généralement pas des experts en méthodes formelles. Des outils appropriés munis d'interfaces attractives et faciles d'emploi doivent être disponibles pour masquer aux concepteurs la complexité de la formalisation. Un environnement de conception devrait permettre à ses utilisateurs de parcourir chaque phase du cycle de développement détaillé dans le Chapitre 3 en incluant une aide à la vérification formelle. Des vues claires de la spécification, de l'architecture et des preuves doivent être générées pour aider le concepteur. Enfin, l'outil doit aussi permettre de revenir en arrière pour ajouter ou modifier des éléments au cours du cycle de développement jusqu'à l'obtention d'une architecture satisfaisante.

L'objet du chapitre suivant est la description de l'outil *CAPRIV* réalisé dans le cadre de cette thèse. Une étude de cas illustre l'utilisation de cet outil dans le chapitre d'après.

CHAPITRE 5

Outil d'aide à la conception

5.1 Introduction

La démarche et le modèle formel présentés précédemment forment une base théorique pour aider le concepteur à construire une architecture répondant aux exigences de protection de la vie privée. Il est cependant difficile de tirer profit d'une formalisation sans le support d'un outil. En effet, le nombre de primitives à manipuler peut vite s'avérer trop important pour se prêter à un traitement manuel. De plus, le risque d'erreur ferait perdre une partie des avantages de la formalisation.

Il est donc nécessaire de disposer d'un outil d'aide à la conception [1]. Cet outil doit :

1. guider le concepteur dans l'application de la démarche systématique présentée en Figure 3.1 du Chapitre 3 ;
2. permettre la vérification des propriétés de l'architecture ainsi conçue au regard de l'axiomatique du Tableau 4.8 du Chapitre 4.

Ces deux points font l'objet des Sections 5.2 et 5.3. Les choix techniques retenus pour l'implémentation seront ensuite détaillés en Section 5.4 et une illustration de l'exemple construit au fil des chapitres précédents est montrée en Section 5.5.

5.2 Interface

L'interface graphique [2] est le moyen par lequel l'utilisateur sera guidé dans la démarche. Les vues successives devront lui permettre de pouvoir avancer et revenir en arrière tel que décrit dans le Chapitre 3.

L'interface graphique doit offrir deux types de fonctionnalités : permettre au concepteur d'agir sur le modèle et visualiser l'effet de ses actions. Ces interactions doivent réduire son exposition au formalisme mathématique sous-jacent sans perdre les avantages qu'il procure.

1. *Computer-Aided Design tool (CAO tool)* en anglais.
2. *Graphical User Interface (GUI)* en anglais.

Cela est permis par l'utilisation de deux vues qui s'affichent simultanément dans l'interface. La modélisation est détaillée dans la Section 5.2.1 et la visualisation en Section 5.2.2.

5.2.1 Modélisation

Pour suivre la démarche systématique, la modélisation se fait au travers de trois onglets qui correspondent aux principales étapes du cycle de développement. L'ordre naturel d'interaction est celui de la démarche, chaque onglet se présentant l'un après l'autre. La navigation entre ces trois onglets est cependant libre et permet au concepteur de revenir sur des choix déjà effectués.

Ces trois onglets sont :

1. **Spécifier** correspond à l'étape de spécification décrite dans la Section 3.4.
2. **Concevoir** correspond à l'étape de conception architecturale décrite dans la Section 3.5.
3. **Vérifier** correspond à l'étape d'évaluation décrite dans la Section 3.6. Nous utilisons le terme « vérifier » plutôt qu'« évaluer » dans l'outil pour insister sur le caractère formel de l'approche [3].

Nous présentons dans la suite le rôle précis de chacun de ces onglets.

Spécifier. La première tâche du concepteur est de déclarer les éléments atomiques (acteurs, variables et fonctions) qui seront utilisés durant le développement. Différentes propriétés de ces éléments, comme le fait qu'une variable est indexée ou qu'une fonction n'est pas inversible, peuvent être sélectionnées à travers le *processus*. Les équations définissant le service sont alors déclarées en utilisant ces éléments. Finalement, les exigences de confidentialité et d'intégrité sont définies à partir des éléments atomiques et du service.

Concevoir. L'étape suivante pour le concepteur est de concevoir l'architecture satisfaisant les exigences spécifiées. À cette fin, le concepteur entre les contraintes pré-existantes et définit les opérations et les types de confiance pour chaque équation du service. Enfin, il peut ajouter les communications manquantes pour la satisfaction du service.

Vérifier. S'il choisit de le faire, le concepteur peut ensuite vérifier l'architecture obtenue. La première vérification concerne la cohérence de l'architecture (par exemple le fait que les arguments d'une opération sont eux-mêmes produits d'une manière ou d'une autre au sein du composant effectuant l'opération). S'il le souhaite, le concepteur peut, enfin, formellement vérifier la satisfaction des exigences de confidentialité et d'intégrité.

5.2.2 Visualisation

La modélisation permet au concepteur de naviguer dans l'espace de conception.

Elle est aussi scindée en trois onglets qui répondent aux trois onglets détaillés dans la section précédente. Chacun affiche le résultat d'une interaction :

3. Nous rappelons que le concepteur peut choisir de suivre la démarche présentée dans le Chapitre 3 de manière informelle.

1. les exigences, pour le résultat de la spécification ;
2. l'architecture, pour le résultat de la conception ;
3. les preuves, pour le résultat de la vérification.

Ces trois vues sont présentées dans les paragraphes qui suivent.

Exigences La vue des exigences affiche tous les éléments atomiques déclarés par le concepteur (ainsi que leurs propriétés particulières). Les équations du service sont aussi listées. Enfin, les exigences de confidentialité et d'intégrité spécifiées sont affichées.

Architecture La vue de l'architecture affiche les primitives architecturales et renseigne le concepteur sur son état.

Preuves La vue des preuves affiche, pour chaque type de vérification (cohérence et exigences de confidentialité et d'intégrité), le résultat de sa vérification et les détails correspondant.

L'interface est le moyen de créer le modèle qui peut ensuite être soumis à la vérification. Cette dernière nécessite l'utilisation d'algorithmes dédiés qui sont présentés dans la section suivante.

5.3 Algorithmes d'inférence

La finalité des algorithmes est de déterminer la véracité d'une exigence écrite dans le langage des propriétés architecturales du Tableau 4.6. La vérification de ces exigences est effectuée en mettant en œuvre les règles du Tableau 4.8.

L'arbre de dépendance des règles montré sur la Figure 5.1indique que ceux-ci forment deux branches distinctes. Celles-ci correspondent exactement aux deux types de propriété, confidentialité en partie gauche et intégrité en partie droite, qui ont été traitées de manière orthogonale dans le cadre formel. Les rectangles arrondis représentent des ensembles d'architecture $Arch$, de dépendances Dep_i, de déductions $\triangleright_i$ et de propriétés de confidentialité et d'intégrité. Les notations Has_i^{all}, Has_i^{one}, Has_i^{none}, K_i, B_i et $\wedge$ représentent les ensembles comprenant toutes les variables et formules vérifiant respectivement $A \vdash Has_i^{all}\left(\tilde{X}\right)$, $A \vdash Has_i^{one}\left(\tilde{X}\right)$, $A \vdash Has_i^{none}\left(\tilde{X}\right)$, $A \vdash K_i\,(Eq)$, $A \vdash B_i\,(Eq)$ et $A \vdash \phi_1 \wedge \phi_2$. Les carrés correspondent aux règles du Tableau 4.8. Ceux-ci permettent d'inférer de nouvelles formules (appartenant à l'ensemble désigné par la flèche sortante) à partir d'autres formules (appartenant aux ensembles desquels viennent les flèches entrantes). Les règles H5, H6, K$\wedge$, K$\triangleright$, B$\wedge$, B$\triangleright$ et I$\wedge$ sont particulières en ce qu'elles utilisent en prémisses des formules appartenant aux mêmes ensembles que la formule de conclusion.

Les algorithmes utilisés pour vérifier les propriétés de ces deux branches sont présentés en Section 5.3.1 pour la confidentialité et en Section 5.3.2 pour l'intégrité.

5.3.1 Confidentialité

La vérification des propriétés de confidentialité peut se faire en calculant les trois ensembles de formules Has_i^{all}, Has_i^{one} et Has_i^{none} puis en vérifiant si la propriété appartient à l'ensemble correspondant.

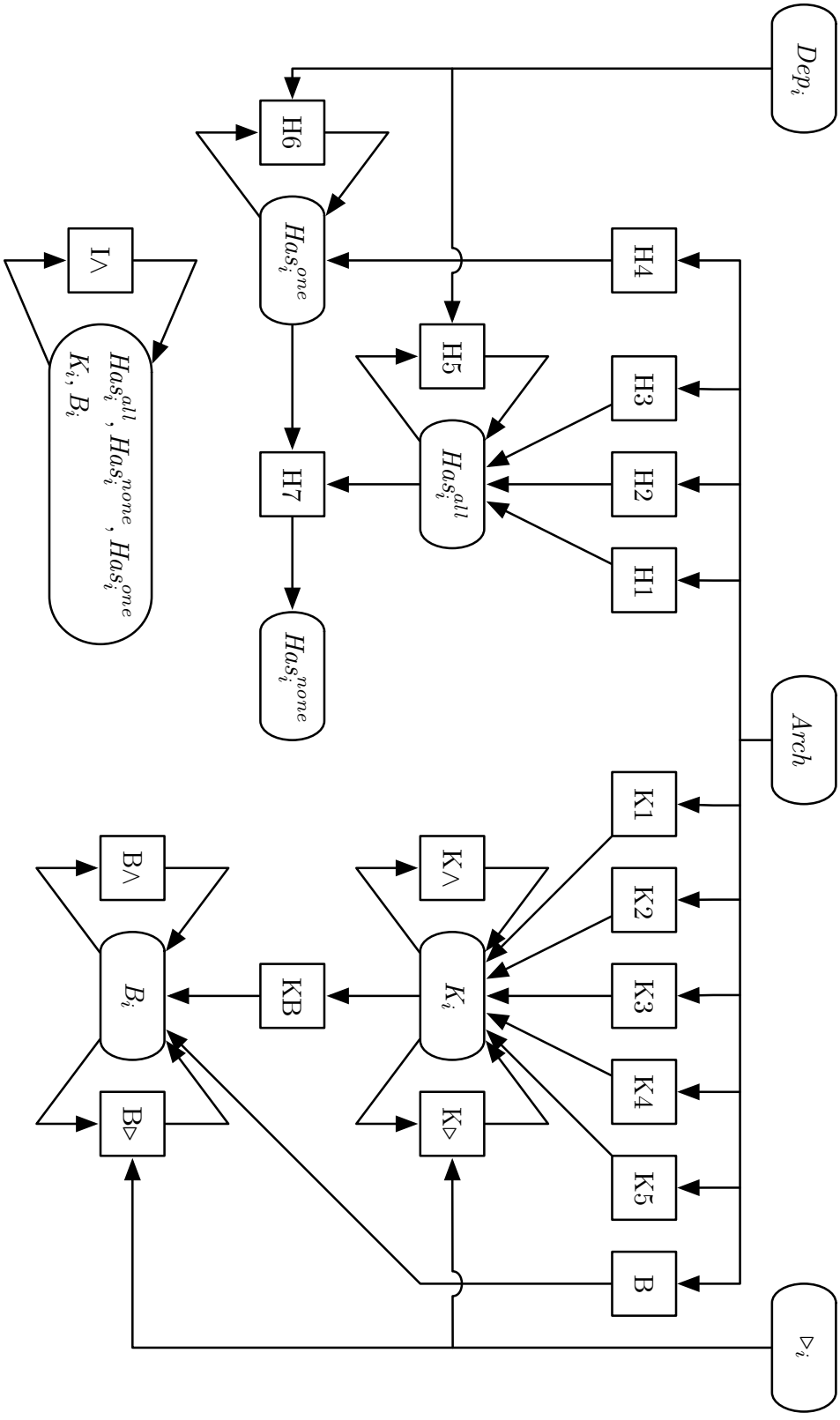

Figure 5.1 – Arbre de dépendance des ensembles de formules $Arch$, Dep_i, $\triangleright_i$, K_i, B_i et $\wedge$ faisant apparaître les règles utiles à leur détermination.

Les Algorithmes 5.1, 5.2 et 5.3 détaillent les procédures pour la construction des ensembles Has_i^{all}, Has_i^{one} et Has_i^{none} respectivement. Par souci de lisibilité, nous avons fait plusieurs simplifications :

- la description des structures de données est de haut niveau : les primitives et variables du langage architectural sont directement utilisées en suivant les conventions du langage des termes (voir Section 4.1), du langage architectural (voir Section 4.2) et de l'axiomatique (voir Section 4.8) ; nous décrivons brièvement dans la Section 5.4 le choix retenu pour leur implémentation ;
- les conditions des boucles `while` sont vérifiées en comparant la taille de l'ensemble courant à sa taille lors de l'itération précédente (ce qui est noté `grows`) ;
- l'appel à `match` est une reconnaissance de motif [4] : les variables libres sont instanciées selon l'élément courant et « _ » désigne des variables anonymes.

Algorithme 5.1 Confidentialité (1 sur 3), calcul de Has_i^{all}.

```
 1: for all i ∈ Comp do                                          ▷ (i ∈ Comp)
 2:     hasAllSet(i) = ∅
 3: end for
 4: for all r ∈ Arch do                                          ▷ (r ∈ A)
 5:     if r match Has(i)(x) then                                ▷ Règle H1 (i ∈ Comp, x ∈ Var)
 6:         hasAllSet(i) ← hasAllSet(i) ∪ {x}
 7:     else if r match Receive(i)(_)(_)(e) then                 ▷ Règle H2 (i ∈ Comp, e ∈ 2^Var)
 8:         for all x ∈ e do                                     ▷ (x ∈ Var)
 9:             hasAllSet(i) ← hasAllSet(i) ∪ {x}
10:         end for
11:     else if r match Compute(g)(x)(_) then                    ▷ Règle H3 (g ∈ 2^Comp, x ∈ Var)
12:         for all i ∈ g do                                     ▷ (i ∈ Comp)
13:             hasAllSet(i) ← hasAllSet(i) ∪ {x}
14:         end for
15:     end if
16: end for
17: for all i ∈ Comp do                                          ▷ (i ∈ Comp)
18:     while size(hasAllSet(i)) grows do                        ▷ Boucle de saturation
19:         for all d ∈ Dep(i) do                                ▷ (d ∈ (Comp × Comp))
20:             if d match (x, y) then                           ▷ (x, y ∈ Vars)
21:                 if y ∈ hasAllSet(i) then                     ▷ Règle H5
22:                     hasAllSet(i) ← hasAllSet(i) ∪ {x}
23:                 end if
24:             end if
25:         end for
26:     end while
27: end for
```

La preuve de correction des Algorithmes 5.1, 5.2 et 5.3 peut désormais être établie.

Preuve de correction. La Figure 5.2 montre l'ordre dans lequel les règles sont appliqués en

4. *Pattern matching* en anglais.

Algorithme 5.2 Confidentialité (2 sur 3), calcul de Has_i^{one}.

for all $i \in Comp$ **do** ▷ $(i \in Comp)$
  $hasOneSet(i) = \emptyset$
end for
for all $r \in Arch$ **do** ▷ $(r \in A)$
  if r **match** $Spotcheck(i)(\_)(x)(\_)$ **then** ▷ Règle H4 $(i \in Comp, x \in Var)$
    $hasOneSet(i) \leftarrow hasOneSet(i) \cup \{x\}$
  end if
end for
for all $i \in Comp$ **do** ▷ $(i \in Comp)$
  while size$(hasOneSet(i))$ **grows do** ▷ Boucle de saturation
    for all $d \in Dep(i)$ **do** ▷ $(d \in (Comp \times Comp))$
      if d **match** (x, y) **then** ▷ $(x, y \in Vars)$
        if $y \in hasOneSet(i)$ **then** ▷ Règle H6
          $hasOneSet(i) \leftarrow hasOneSet(i) \cup \{x\}$
        end if
      end if
    end for
  end while
end for

Algorithme 5.3 Confidentialité (3 sur 3), calcul de Has_i^{none}.

for all $i \in Comp$ **do** ▷ $(i \in Comp)$
  $hasNoneSet(i) = \emptyset$ ▷ ($hasAllSet(i)$ et $hasOneSet(i)$ déjà définis)
end for
for all $i \in Comp$ **do** ▷ $(i \in Comp)$
  for all $x \in Var$ **do** ▷ $(x \in Var)$
    if $x \notin (hasAllSet(i) \cup hasOneSet(i))$ **then** ▷ Règle H7
      $hasNoneSet(i) \leftarrow hasNoneSet(i) \cup \{x\}$
    end if
  end for
end for

partant de trois ensembles initialement vides $hasAllSet(i)$, $hasOneSet(i)$ et $hasNoneSet(i)$ représentant respectivement les ensembles Has_i^{all}, Has_i^{one} et Has_i^{none}. Cet ordre est compatible avec les relations de dépendance entre les ensembles (voir Figure 5.1).

Ces ensembles sont ensuite augmentés des propriétés correspondant aux règles applicables [5] dans l'ordre mentionné. Les règles H5 et H6, qui font intervenir en prémisses les ensembles eux-mêmes, sont appliquées jusqu'à saturation de ces ensembles.

L'ensemble Has_i^{all} est complètement défini par application des règles H1, H2, H3 et H5 puisqu'aucune autre règle ne permet d'inférer une propriété du type $A \vdash Has_i^{all}\left(\tilde{X}\right)$. L'ensemble Has_i^{one}, quant à lui, est complètement défini par l'application des règles H4 et H6. Enfin,

5. Nous écrivons « règles applicables » plutôt que « instances possibles des règles applicables » par souci de lisibilité.

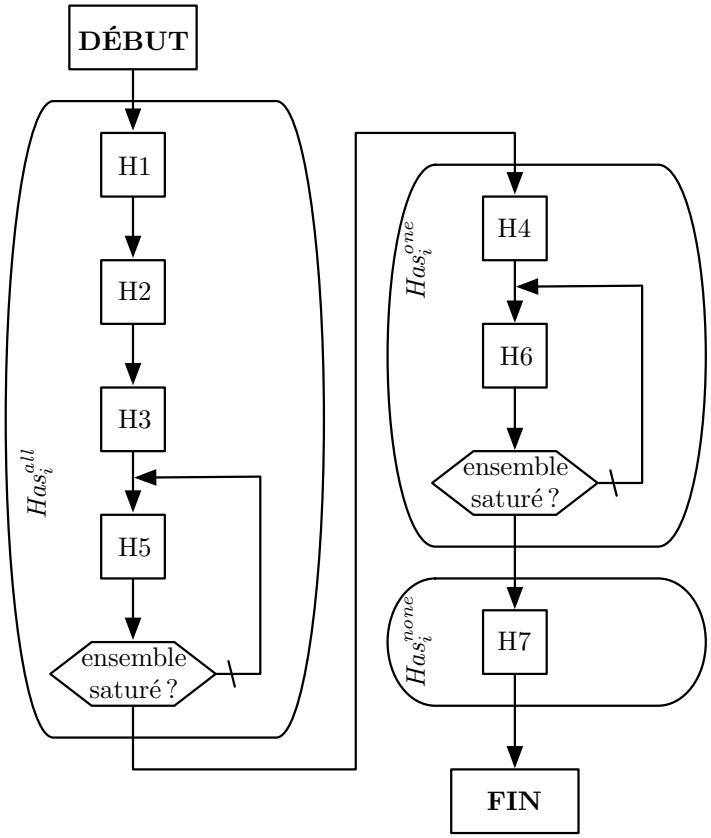

FIGURE 5.2 – Algorigramme de calcul des ensembles de formules Has_i^{all}, Has_i^{one} et Has_i^{none}.

l'ensemble Has_i^{none} est complètement défini par application de la règle H7.

Les ensembles Has_i^{all} puis Has_i^{one} et enfin Has_i^{none} sont ainsi construits.

Les règles s'appliquent en testant l'appartenance de leurs prémisses aux ensembles représentant l'architecture *Arch* et les dépendances Dep_i (par reconnaissance de motifs). Si les prémisses sont vérifiées, alors la conclusion est ajoutée dans l'un des ensembles représentant les propriétés. Les règles H5 et H6 sont appliquées par saturation des ensembles correspondants. Chaque ensemble contient alors les variables satisfaisant la propriété correspondante. L'algorithme est donc correct. □

La preuve de terminaison de ces trois algorithmes est maintenant donnée.

Preuve de terminaison. Les ensembles sur lesquels itèrent les boucles de l'Algorithme 5.1 sont *Comp* pour les lignes 1, 12 et 17, *Arch* pour la ligne 4, *Var* pour la ligne 8, $(Comp \times Comp)$ pour la ligne 19 et *Vars* pour la ligne 20. Pour l'Algorithme 5.2, les boucles se font pour les lignes 1 et 9 sur *Comp*, pour la ligne 4 sur *Arch*, pour la ligne 10 sur $hasAllSet(i) \subseteq Vars$ et pour la ligne 11 sur $(Comp \times Comp)$. Enfin, l'Algorithme 5.3 boucle aux lignes 1 et 4 sur *Comp* et à la ligne 5 sur *Vars*.

Or *Comp* et *Vars* sont les ensembles des composants et des variables qui sont finis par hypothèse. L'ensemble *Arch* contient l'architecture courante qui est aussi finie par hypothèse. Enfin, les opérations ensemblistes d'union $\cup$, de produit cartésien $\times$ et d'ensemble des parties d'un ensemble $\mathcal{P}$ (notées 2^B dans les commentaires des algorithmes) sont appliquées à des ensembles finis et produisent donc des ensembles finis. Toutes les boucles mentionnées précédemment itèrent donc sur des ensembles finis.

Enfin, la boucle de saturation sur $hasAllSet(i)$ se termine car la taille de $hasAllSet(i)$ possède un majorant qui est $|Vars|$: la boucle effectue donc au maximum autant d'itérations (plus une pour la comparaison). Par conséquent, la procédure elle-même se termine. □

Les algorithmes d'inférence pour les propriétés d'intégrité, objet de la section suivante reposent sur le même schéma. Les relations de déduction $\triangleright_i$ demandent cependant un traitement particulier.

5.3.2 Intégrité

La vérification des propriétés d'intégrité peut se faire comme dans la section précédente en calculant les deux ensembles K_i et B_i. Il peut toutefois s'avérer que ce calcul ne puisse être que partiel car les relations de déduction $\triangleright_i$ sont construites à partir des termes (voir Tableau 4.1) qui ne sont pas finis *a priori* (en raison de F). Dans ce cas, les algorithmes d'application des règles K$\triangleright$ et B$\triangleright$ ne seront que semi-décidables.

En effet, celles-ci reposent sur des égalités entre termes. La logique du premier ordre avec égalité est connue pour n'être que semi-décidable dès lors qu'elle comporte plusieurs fonctions au moins unaires [Mon76].

Les procédures sont détaillées dans les Algorithmes 5.4 et 5.5. Nous avons toutefois écarté les règles K$\triangleright$, K$\wedge$, B$\triangleright$, B$\wedge$ et I$\wedge$ pour les raisons évoquées (K$\wedge$, B$\wedge$ et I$\wedge$ dépendant des résultats de K$\triangleright$ et B$\triangleright$ comme montré dans la Figure 5.1). Nous nommons K'_i et B'_i ces ensembles partiels.

Nous établissons maintenant les preuves de correction et de terminaison des Algorithmes 5.4 et 5.5.

Preuve de correction. La Figure 5.3 définit un ordre dans lequel les règles sont appliquées. Cet ordre est compatible avec les relations de dépendance entre les ensembles (voir Figure 5.1).

L'ensemble K'_i est défini par application des règles K1, K2, K3, K4 et K5.

L'ensemble B'_i est construit par application des règles B et KB. Il ne peut donc être construit qu'après la construction de l'ensemble K'_i car il dépend de celui-ci à travers l'application de la règle KB.

Les règles sont appliquées comme décrit dans la preuve de correction précédente. Les algorithmes sont donc corrects. □

Preuve de terminaison. Ces deux algorithmes se terminent pour les mêmes raisons que les algorithmes construisant les ensembles relatifs à la confidentialité : les ensembles d'itération

Algorithme 5.4 Intégrité (1 sur 2), calcul de K'_i.

for all $i \in Comp$ **do** ▷ $(i \in Comp)$
  $kSet(i) = \emptyset$
end for
for all $r \in Arch$ **do** ▷ $(r \in A)$
  if r **match** $Compute(g)(x)(t)$ **then** ▷ Axiome K1 $(g \in 2^{Comp}, x \in Var, t \in T)$
    for all $i \in g$ **do** ▷ $(i \in Comp)$
      $kSet(i) \leftarrow (x = t)$
    end for
  else if r **match** $Check(i)(e)$ **then** ▷ Axiome K2 $(i \in Comp, e \in 2^{Eq})$
    for all $eq \in e$ **do** ▷ $(eq \in Eq)$
      $kSet(i) \leftarrow eq$
    end for
  else if r **match** $VerifProof(i)(\_)(e)$ **then** ▷ Axiome K3 $(i \in Comp, e \in 2^{Eq})$
    for all $eq \in e$ **do** ▷ $(eq \in Eq)$
      $kSet(i) \leftarrow eq$
    end for
  else if r **match** $VerifProof(i)(\_)(e)$ **then** ▷ Axiome K4 $(i \in Comp, e \in 2^{Eq})$
    for all $r' \in e$ **do** ▷ $(r' \in Pro)$
      if r' **match** $Attest(k)(e')$ **then** ▷ $(k \in Comp, e' \in Att)$
        for all $r'' \in Arch$ **do** ▷ $(r'' \in A)$
          if r'' **match** $Trust(i')(k')$ **then** ▷ $(i' \in Comp, k' \in Comp)$
            if $i' == i$ **and** $k' == k$ **then**
              for all $eq \in e'$ **do** ▷ $(eq \in Eq)$
                $kSet(i) \leftarrow eq$
              end for
            end if
          end if
        end for
      end if
    end for
  else if r **match** $VerifAttest(i)(j)(e)$ **then** ▷ Axiome K5 $(i \in Comp, j \in Comp, e \in 2^{Eq})$
    for all $r' \in Arch$ **do** ▷ $(r' \in A)$
      if r' **match** $Trust(i')(j')$ **then** ▷ $(i' \in Comp, j' \in Comp)$
        if $i' == i$ **and** $j' == j$ **then**
          for all $eq \in e'$ **do** ▷ $(eq \in Eq)$
            $kSet(i) \leftarrow eq$
          end for
        end if
      end if
    end for
  end if
end for

Algorithme 5.5 Intégrité (2 sur 2), calcul de B'_i.

```
1:  for all i ∈ Comp do                                   ▷ (i ∈ Comp)
2:      bSet(i) = ∅                                       ▷ (kSet(i) déjà définis)
3:  end for
4:  for all r ∈ Arch do                                   ▷ (r ∈ A)
5:      if r = Spotcheck(i)(_)(_)(e) then                 ▷ Axiome B (i ∈ Comp, e ∈ 2^Eq)
6:          for all eq ∈ e do                             ▷ (e ∈ Eq)
7:              bSet(i) ← eq
8:          end for
9:      end if
10: end for
11: for all i ∈ Comp do                                   ▷ (i ∈ Comp)
12:     for all eq ∈ kSet(i) do                           ▷ Axiome KB (eq ∈ Eq)
13:         bSet(i) ← eq
14:     end for
15: end for
```

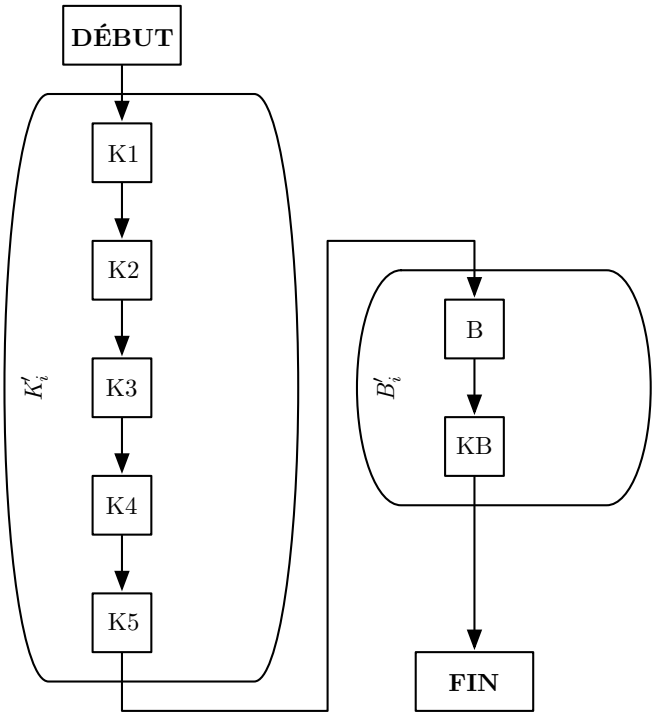

Figure 5.3 – Algorigramme de création des ensembles de formules K'_i et B'_i.

sont finis (les ensembles *Pro*, *Att* et *Eq* sont définis par l'architecture, elle-même finie par hypothèse). □

Remarque. L'ensemble B_i ne devrait commencer à être construit qu'une fois l'ensemble K_i terminé. Néanmoins, cela peut s'avérer impossible dans le cas d'une relation de déduction $\triangleright_i$ non décidable. Cependant, nous pouvons remarquer que les règles K$\triangleright$ et K$\wedge$ d'une part, et B$\triangleright$ et B$\wedge$ d'autre part, sont identiques à la modalité (connaissance ou croyance) près. Par conséquent, pour déduire une croyance, il est toujours possible d'appliquer les règles K1 à K5, puis KB et B avant d'appliquer B$\triangleright$ et B$\wedge$ (plutôt que d'appliquer K$\triangleright$ avant KB).

Ce dernier point est particulièrement utile pour l'implémentation de l'outil car il évite d'avoir à faire appel à deux procédures indécidables pour la vérification des propriétés de croyance. Nous faisons appel à un prouveur de théorème tiers pour l'application des règles impliquant les $\triangleright_i$.

5.4 Implémentation

Les algorithmes présentés dans le chapitre précédent ont été implémentés dans un outil d'aide à la conception nommé *CAPRIV* [6]. Celui-ci est un prototype et comporte à ce stade certaines limitations dans l'interface.

CAPRIV a été développé en Scala [OSV10] qui est un langage fonctionnel (impur) typé orienté objet. Le programme est exécuté sur la machine virtuelle Java [7], ce qui permet d'être indépendant de la plate-forme et d'utiliser les bibliothèques Java (comme Swing qui a été utilisé pour construire l'interface graphique). L'utilisation de Scala simplifie l'implémentation selon l'approche *LCF* [8] [Gor00] qui garantit statiquement que les théorèmes ne peuvent être déduits que des axiomes et de l'application des règles d'inférence. Tous les éléments du cadre formel (langages et éléments atomiques) ont été modélisés sous la forme de types de données algébriques. Ceux-ci ont été implémentés dans Scala sous la forme de classes abstraites. La technique de reconnaissance de motifs [9] rend l'utilisation de ces classes abstraites naturelle. Le résultat est un code plus concis et lisible que ce qui aurait été possible d'obtenir en Java.

La vérification de certaines des propriétés de connaissance et de croyance repose sur des prouveurs de théorème [10] et *CAPRIV* agit dans ce cas comme un point d'entrée [11] vers *Why3* [Bob+11]. *Why3* est une plateforme dans laquelle des théories peuvent être exprimées dans un langage proche de *ML* [12]. Des bibliothèques standards sont proposées pour modéliser facilement des structures telles que des anneaux ou des tableaux par exemple. *Why3* repose lui-même sur des prouveurs de théorème tiers tels que Alt-Ergo [Bob+08] ou Z3 [DMB08] entre autres.

6. *Computer-Aided PRIVacy*.
7. *Java Virtual Machine* en anglais.
8. *Logic for Computable Functions*.
9. *Pattern matching* en anglais.
10. *Automatic Theorem Provers* en anglais.
11. *Front-end* en anglais.
12. *MetaLanguage*.

CAPRIV génère automatiquement des théories correspondant aux architectures (sous la forme d'axiomes et de conjectures à prouver), appelle *Why3* et traite sa réponse à travers son interface de programmation [13]. Par exemple, si la fonction H est injective, alors l'axiome :

```
axiom h_inj : forall x y : t. ((h x = h y) -> (x = y))
```

est ajouté dans la théorie. Si la fonction HH admet une propriété d'homomorphisme, alors l'axiome :

```
axiom hh_hom : forall x : t. ((mult (hh x)) = (hh (add x)))
```

est ajouté dans la théorie pour les variables tableaux.

Quand une propriété attendue ne peut être prouvée, un concepteur suffisamment expert peut choisir d'utiliser directement l'interface de *Why3* qui peut être appelée depuis *CAPRIV*. Il peut alors s'appuyer sur les possibilités de preuve interactive de théorèmes de *Why3* en essayant plusieurs stratégies. Il est même possible pour un concepteur expert d'exploiter les configurations détaillées ou directement les théories — ces compétences ne sont toutefois pas requises pour un usage standard de *CAPRIV*. Le choix de se reposer sur un outil externe tel que *Why3* montre qu'il est possible d'intégrer le développement de solutions respectueuses de la vie privée dès la conception dans des cadres déjà existant. Ce dernier point est particulièrement important pour la diffusion des outils et des pratiques.

5.5 Exemple d'utilisation de *CAPRIV*

Nous reprenons l'exemple du relevé intelligent de consommation électrique commencé dans les chapitres précédents. Le but est de concevoir une architecture répondant à des exigences de confidentialité fortes vis-à-vis du fournisseur d'électricité qui ne doit avoir accès qu'au minimum possible de données à caractère personnel. Le schéma architectural est présenté en Figure 4.1.

Plusieurs exemples d'interaction avec *CAPRIV* correspondant à chacune des étapes du développement décrit en Section 5.2.1 sont présentés dans les sections suivantes : pour la spécification des exigences, la conception architecturale et la vérification.

Spécification. La première étape est la spécification des exigences. Le résultat obtenu par l'utilisation de l'outil *CAPRIV* est représenté en Figure 5.4.

Le concepteur rentre tous les éléments atomiques et construit les équations du service et les exigences à partir de cet onglet. Les résultats apparaissent directement. Les éléments apparaissent dans des menus déroulants afin de minimiser le risque d'erreur de saisie (principe de la non-répétition [14]).

Conception. L'étape suivante est la conception architecturale. La Figure 5.5 montre une capture d'écran de l'outil *CAPRIV* à cette étape.

13. *Application Programming Interface (API)* en anglais.
14. *Don't Repeat Yourself (DRY)* en anglais.

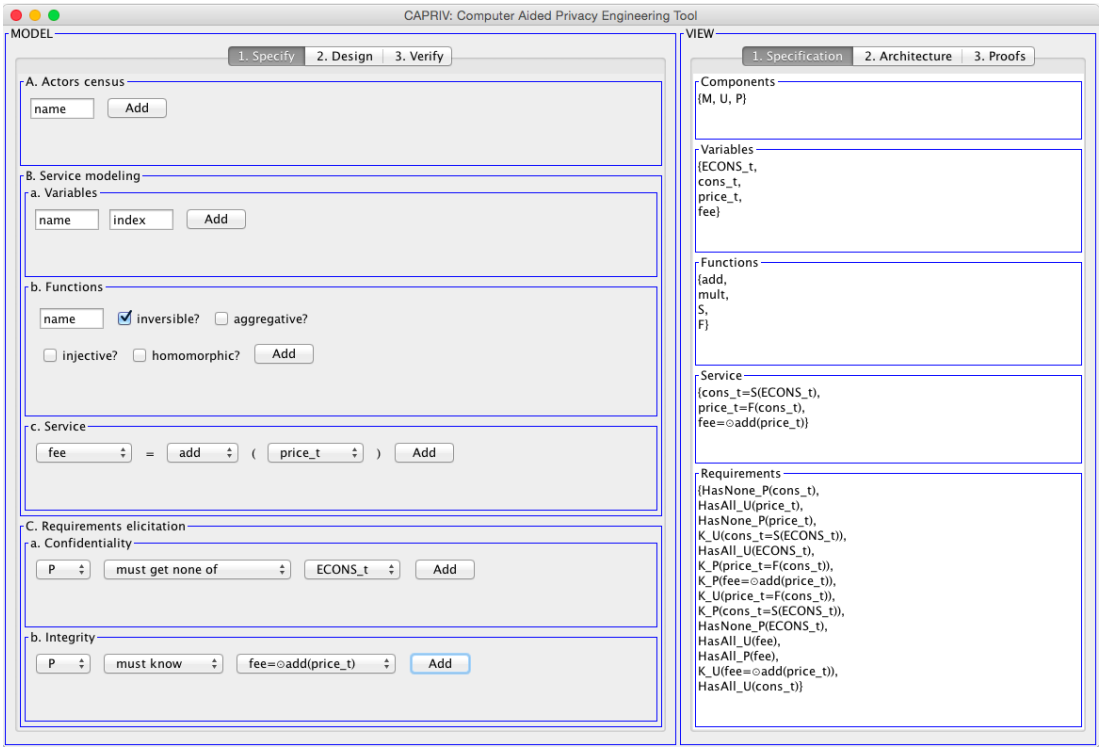

FIGURE 5.4 – Phase de spécification dans *CAPRIV* pour Ω_1.

Cet onglet fait apparaître de manière très ordonnée le cycle de conception. Plus aucun élément ne peut être entré par le clavier à cette étape. Après avoir entré les contraintes, le concepteur peut choisir une localisation et un type de confiance par composant concerné pour chaque opération. Enfin, les communications de variables et de primitives peuvent être sélectionnées pour finaliser l'architecture.

Vérification. L'étape suivante est la vérification des exigences. Nous prenons deux exemples dont les résultats attendus ont déjà été calculés manuellement dans le chapitre précédent.

Une exigence qu'il est important de vérifier, issue du Tableau 4.11, est le non-accès de l'opérateur à la consommation détaillée de l'usager : $Has_{\mathrm{P}}^{none}\ (ECONS_t)$. Une capture d'écran de *CAPRIV* pour la vérification de cette propriété est donnée en Figure 6.6. Cette vérification correspond à ce qui était attendu avec l'utilisation de la règle (H7).

La seconde vérification porte sur une exigence d'intégrité passée en revue dans le Tableau 4.12. Le concepteur souhaite vérifier si l'usager peut être convaincu de la correction du montant final de la facture qui lui est soumis par rapport aux prix intermédiaires. *CAPRIV* fait alors appel à la plateforme *Why3*. Une capture d'écran de *Why3* avec une théorie générée par *CAPRIV* pour la vérification de la propriété $K_{\mathrm{P}}\ (fee = \odot + price_t)$ est donnée en Figure 5.7. Les axiomes de cette théorie correspondent aux relations dérivées sans utilisation de (K▷). Le résultat est aussi conforme à ce qui était attendu et cohérent avec les développements des chapitres précédents.

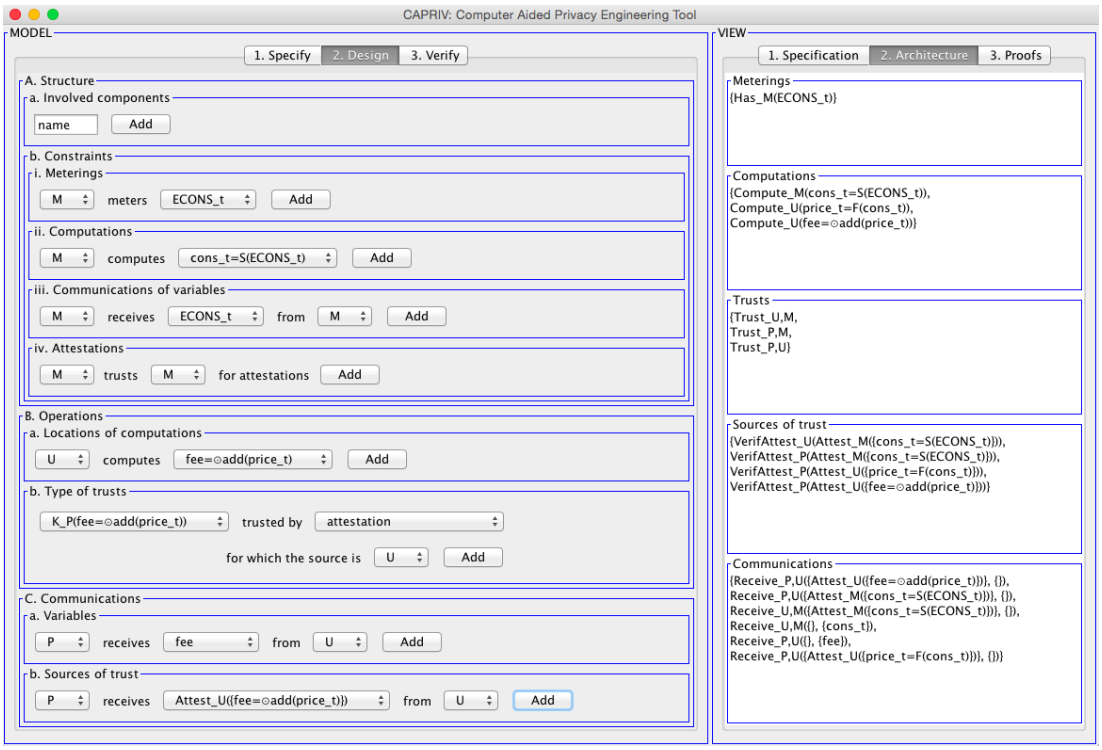

Figure 5.5 – Phase de conception dans *CAPRIV* pour Ω_1.

5.6 Conclusion

L'outil *CAPRIV* présenté dans ce chapitre facilite la tâche du concepteur en le guidant à travers la démarche systématique détaillée dans le Chapitre 3. Cette aide à la conception d'architectures lui permet de bénéficier du formalisme défini dans le Chapitre 4 sans y être directement confronté grâce à l'interface graphique. Il peut donc vérifier que les exigences sont satisfaites directement dans *CAPRIV* et s'appuyer sur des outils standards en cas de besoin.

Une étude de cas où plusieurs types de confiance sont explorés est présentée dans le chapitre suivant. L'impact de ces types de confiance sur l'architecture pourra ainsi être analysé. L'apport de l'outil sera illustré au fil de la démarche en poursuivant l'exemple.

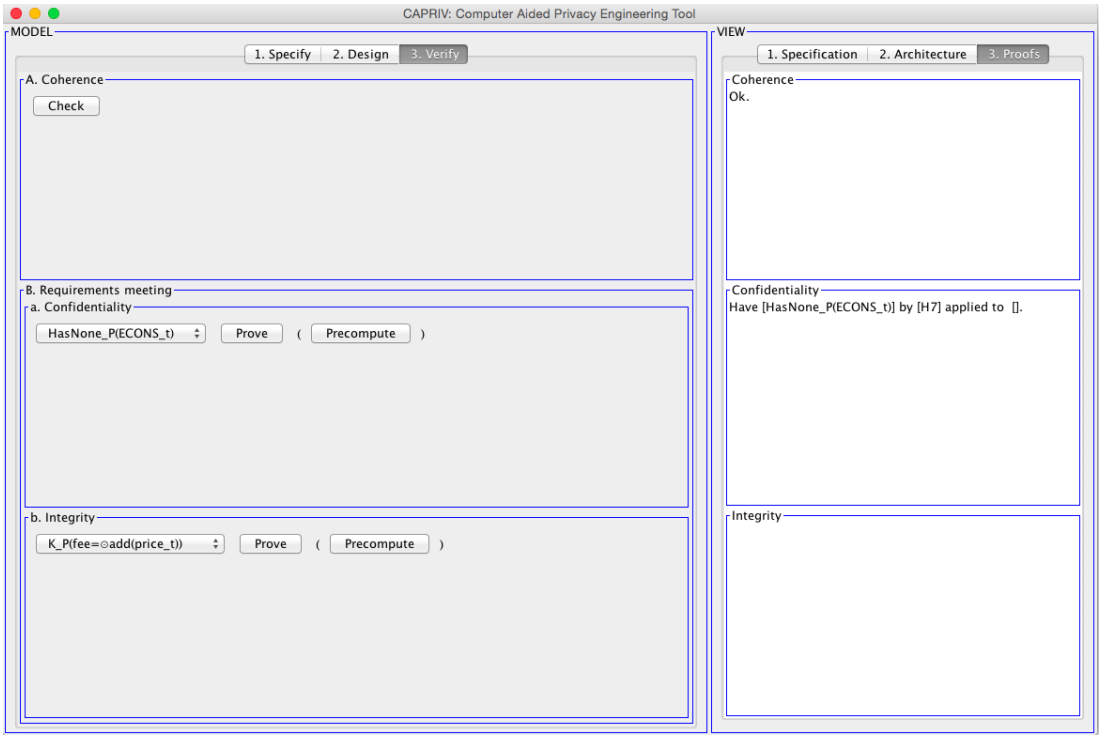

FIGURE 5.6 – Vérification de Has_{P}^{none} $(ECONS_t)$ dans *CAPRIV* pour Ω_1.

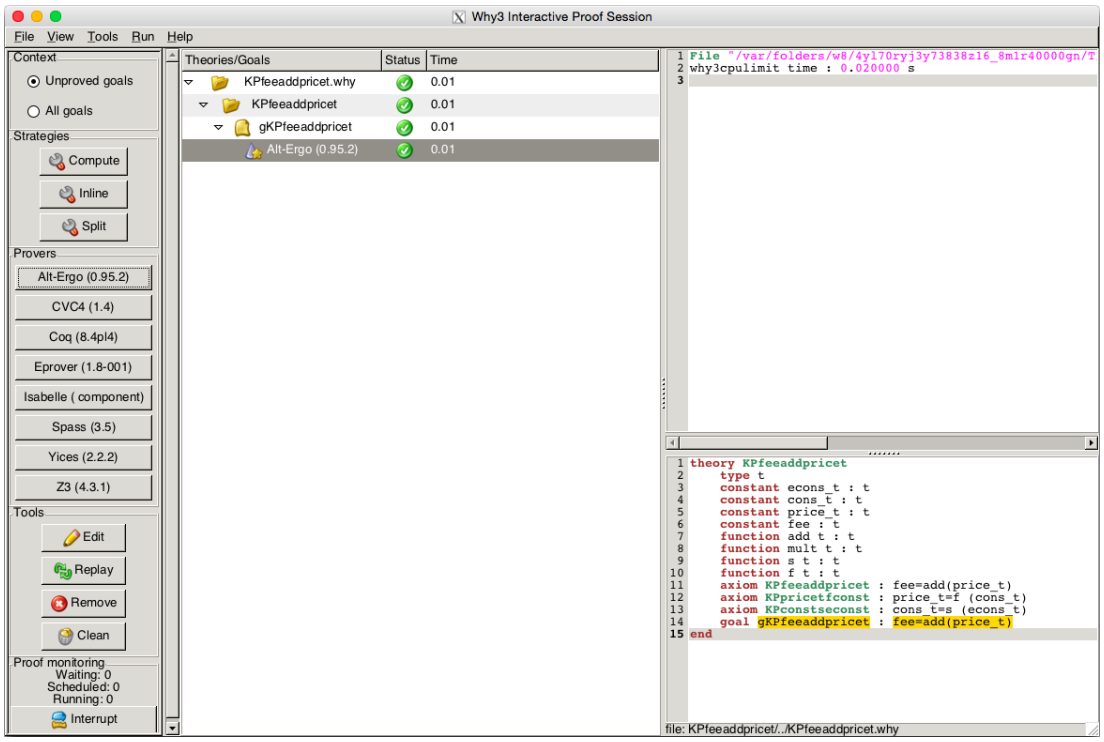

Figure 5.7 – Vérification de K_P ($fee = \odot + price_t$) dans *Why3* pour Ω_1.

CHAPITRE 6

Cas d'étude

6.1 Introduction

L'outil présenté dans le chapitre précédent guide un concepteur non-expert dans la démarche systématique. Il permet aussi de vérifier formellement la satisfaction des exigences de confidentialité et d'intégrité. Ce chapitre propose d'étendre le cas d'étude détaillé au cours des chapitres précédents.

Cette étude permet d'illustrer que :

1. la démarche systématique proposée dans le Chapitre 3 aide le concepteur à concevoir une architecture respectueuse de la vie privée des utilisateurs ;
2. l'outil présenté dans le Chapitre 5 facilite le suivi de cette démarche ;
3. l'outil permet la vérification formelle des exigences de confidentialité et d'intégrité selon les définitions du Chapitre 4.

Le relevé intelligent de consommation électrique est une application industrielle qui peut impliquer des relevés de consommation très fréquents s'avérant intrusifs pour les usagers (voir [Liu+12]). Différentes sortes de services peuvent être fournis dans de tels systèmes comme la facturation de la consommation ou la surveillance de la charge du réseau électrique. Nous explorons dans cette étude le cas de la facturation.

Trois solutions architecturales reposant sur des types de confiance différents sont spécifiées, proposées et évaluées dans les Sections 6.2 pour la confiance par attestation [1], 6.3 pour la confiance par sécurité et 6.4 pour la confiance par redevabilité.

6.2 Première solution : confiance par attestation

La première solution que nous décrivons est principalement basée sur la confiance par attestation. Le compteur M est chargé d'effectuer les mesures et l'usager U est en charge de la

1. Cette première solution est une reprise compacte et continue de l'exemple détaillé au fil des chapitres précédents.

tarification et du calcul du montant *fee*. Tous ces calculs sont attestés par les composants qui en sont les auteurs.

La Figure 6.1 montre une représentation informelle de l'architecture envisagée [2]. Nous donnons cette figure en amont de la démarche par souci de clarté. En situation réelle, le concepteur d'un système peut avoir une idée de l'architecture avant d'appliquer la démarche systématique ou s'appuyer sur la démarche pour construire une architecture par application successive des différentes étapes.

Nous commençons par la phase de spécification, dans la Section 6.2.1 avant de concevoir l'architecture dans la Section 6.2.2. La phase de vérification est ensuite appliquée dans la Section 6.2.3.

6.2.1 Spécification

Nous suivons la démarche de spécification systématique détaillée dans la Section 3.4 en commençant par recenser les acteurs dans la Section 6.2.1.1. Le service est ensuite modélisé dans la Section 6.2.1.2 et les exigences sont définies en Section 6.2.1.3.

6.2.1.1 Recensement des acteurs

Trois acteurs sont nécessaires pour modéliser le service. Les usagers sont naturellement les sujets des données personnelles relatives à leur propre consommation électrique et le fournisseur d'électricité agit comme responsable de traitement pour calculer le prix à facturer à l'usager à la fin de la période de facturation (voir Section 1.1.3). Enfin, les relevés de consommation sont effectués par des compteurs.

L'usager est désigné par $\mathrm{U} \in \mathcal{DS}$ avec U un identifiant unique. Le fournisseur est désigné par $\mathrm{P} \in \mathcal{DC}$ et le compteur par $\mathrm{M} \in \mathcal{DP}$.

6.2.1.2 Modélisation du service

Dans notre exemple, le service se modélise comme un montant à facturer, qui est la somme des tarifs correspondant aux consommations à chaque instant.

Afin d'exprimer les unités de temps, nous introduisons l'indice t borné par le nombre de périodes temporelles retenues dans la période de facturation. Nous pouvons alors modéliser le service Ω_1 comme suit pour l'usager U :

$$\Omega_1 = \left\{ \begin{array}{c} fee = \sum_t (price_t) \\ price_t = \mathrm{F}\,(cons_t) \\ cons_t = \mathrm{S}\,(ECONS_t) \end{array} \right\}$$

avec F une politique tarifaire permettant d'obtenir le tarif price_t pour une consommation particulière cons_t de l'acteur U à l'instant t. La fonction S représente la mesure de la consommation physique réelle $ECONS_t$. La version graphique de ce service est représentée en Figure 6.2.

2. Comme dans le Chapitre 3, les flèches en pointillés représentent les liens de confiance par attestation. Les flèches pleines représentent les envois d'un composant à un autre. Ceux-ci peuvent consister en des variables et des déclarations. Enfin, les équations dans les composants représentent la localisation des calculs.

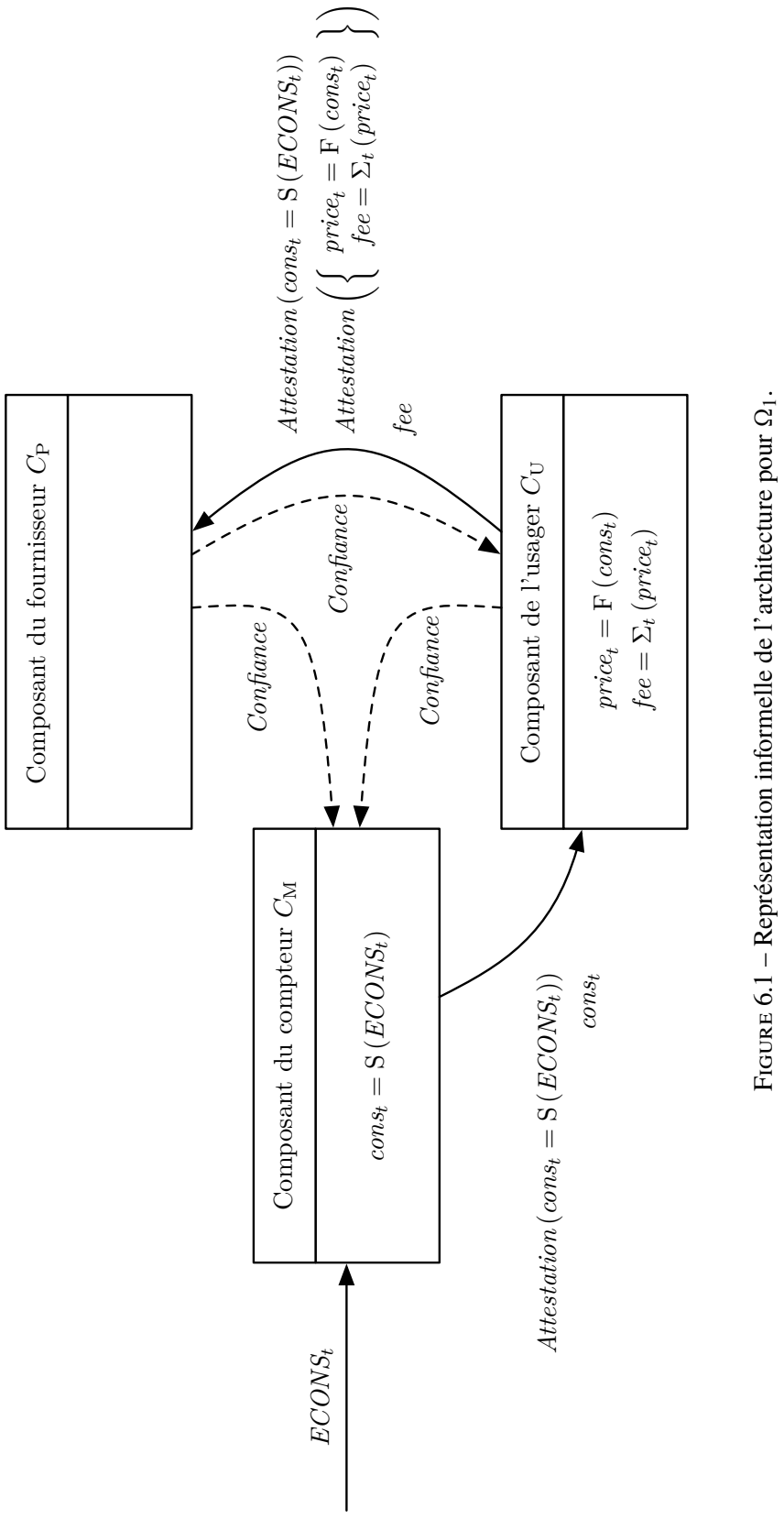

FIGURE 6.1 – Représentation informelle de l'architecture pour Ω_1.

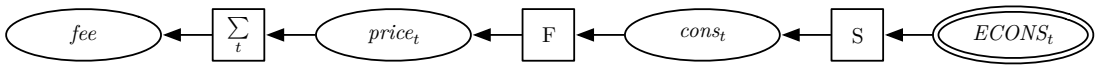

Figure 6.2 – Modèle du service Ω_1.

Tous les acteurs, à savoir l'usager U, le fournisseur d'électricité P et le compteur M connaissent le service Ω_1 et peuvent s'en servir pour calculer ou raisonner.

Dans cet exemple, on peut noter que $ECONS_t$ est une variable de l'environnement. Nous supposons que $\sum$ est une fonction agrégative et que F et S sont des fonctions génériques supposées facilement inversibles par défaut.

L'étape suivante est la définition des exigences à partir du modèle Ω_1.

6.2.1.3 Définition des exigences

Dans notre exemple, à la fois l'usager U et le fournisseur d'électricité P doivent avoir accès aux montants *fee* concernant l'usager. De plus, U doit avoir accès à ses données personnelles $ECONS_t$, $cons_t$ et $price_t$. En revanche, le fournisseur d'électricité P ne doit pouvoir accéder ni aux variables $ECONS_t$, ni aux $cons_t$, ni aux $price_t$ car les fonctions de mesure S et de tarification F, connues de tous, sont facilement inversibles. Ces exigences de confidentialité sont résumées dans le Tableau 6.1.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	U	P
$cons_t$	U	P
$price_t$	U	P
fee	U, P	

Tableau 6.1 – Spécification des exigences de confidentialité pour Ω_1.

Pour ce qui est de l'intégrité, les usagers comme le fournisseur doivent avoir confiance dans la véracité des relations $fee = \sum_t (price_t)$, $price_t = \mathrm{F}(cons_t)$ et $cons_t = \mathrm{F}(ECONS_t)$ (sans que cela implique que le fournisseur ait accès aux variables $ECONS_t$, $cons_t$ et $price_t$). Ces exigences d'intégrité sont récapitulées dans le Tableau 6.2.

Relations	Connaissance
$fee = \sum_t (price_t)$	U, P
$price_t = \mathrm{F}(cons_t)$	U, P
$cons_t = \mathrm{S}(ECONS_t)$	U, P

Tableau 6.2 – Spécifications des exigences d'intégrité pour Ω_1.

La définition des exigences clôt cette phase de la démarche. Le résultat obtenu par l'utilisation de l'outil *CAPRIV* est représenté en Figure 6.3.

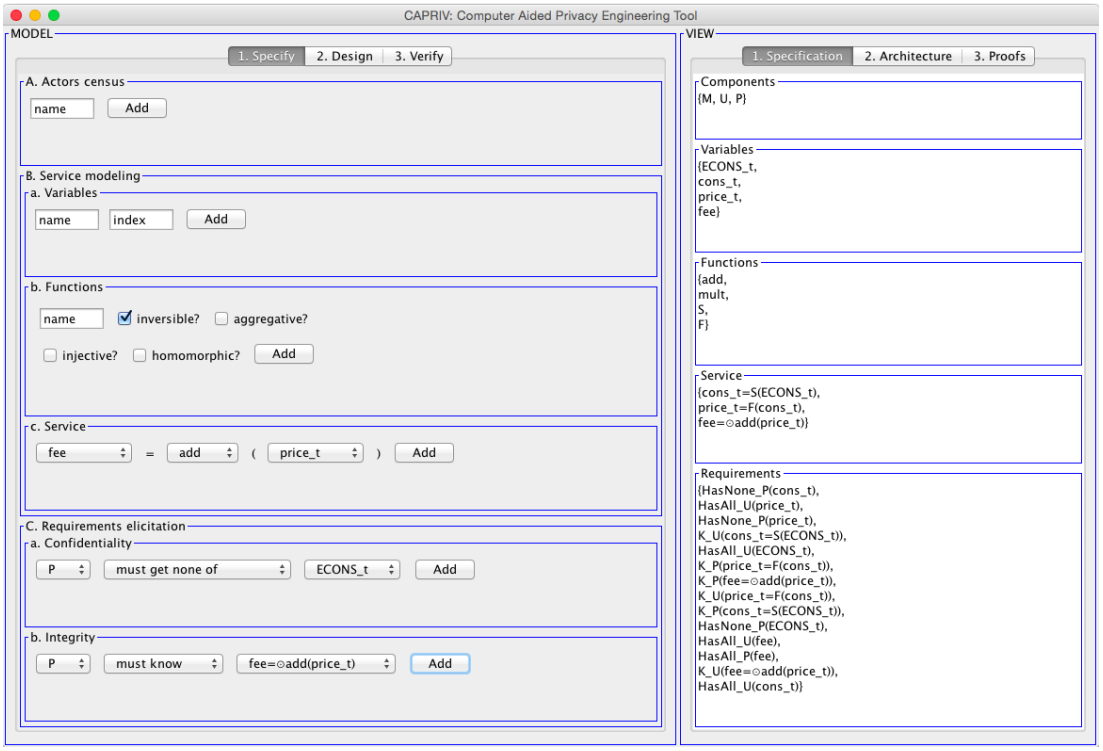

FIGURE 6.3 – Phase de spécification dans *CAPRIV* pour Ω_1.

Une fois la spécification décrite, le concepteur passe à la phase de conception architecturale. Nous décrivons cette étape dans la section suivante.

6.2.2 Conception architecturale

Le concepteur conçoit une architecture qui doit satisfaire la spécification définie dans la section précédente.

Cette phase de conception suit les trois étapes de la démarche systématique présentées dans la Section 3.5 que sont la structuration en Section 6.2.2.1, l'opérationnalisation en Section 6.2.2.2 et la finalisation en Section 6.2.2.3.

6.2.2.1 Structuration

La structuration pose les bases les plus importantes de l'architecture. Les choix effectués à cette étape ont des conséquences sur toute la suite de la démarche de conception.

L'étude des composants participants, des choix imposés et de l'anonymat au sein de l'architecture composent cette première étape en suivant la démarche détaillée dans la Section 3.5.1.

Composants participants. Dans notre exemple, nous considérons que chaque acteur est représenté par un composant dans l'architecture. Le système comprend donc un composant

C_M pour le compteur, un composant C_U pour l'usager et un composant C_P pour le fournisseur d'électricité [3].

Choix imposés. Les $ECONS_t$ sont les grandeurs qui doivent être mesurées. L'opération de mesure, désignée par la fonction S, doit être effectuée par le compteur C_M. Ces compteurs ne doivent avoir accès qu'aux données qu'ils mesurent, soit les $ECONS_t$.

Anonymat. Le service de facturation de notre exemple est par définition nominatif. Il ne comporte donc pas d'exigence d'anonymat.

Une fois la structure de l'architecture définie dans cette première étape, la conception se poursuit par la phase d'opérationnalisation.

6.2.2.2 Opérationnalisation

La phase d'opérationnalisation permet au concepteur de choisir où sont effectués les calculs. Les types de confiance des différents acteurs pour ces calculs sont aussi choisis dans cette phase.

La localisation des calculs, l'éventuelle sélection d'affaiblissements et le choix des types de confiance sont les étapes majeures de cette phase suivant la démarche détaillée dans la Section 3.5.2.

Localisation des calculs. La première opération est la fonction de mesure S. Celle-ci a déjà été localisée dans le compteur dans la phase précédente (il s'agit d'un choix imposé).

La deuxième fonction rencontrée dans la spécification de notre exemple est F. Comme les données de consommation sont déjà connues du compteur C_M, l'option privilégiée consiste à localiser le calcul F au même endroit. Cependant, cette hypothèse n'est pas forcément réaliste car les compteurs ont en général des capacités de calcul limitées et ne servent qu'une seule fonction qui est celle de la mesure (S ici). La deuxième option pour effectuer ce calcul est alors le composant C_U contrôlé par l'usager U (en pratique, il peut s'agir de son *PC* ou d'un équipement dédié). C'est l'option du calcul par le composant de l'usager C_U qui est retenue.

La dernière fonction apparaissant dans la spécification est la somme $\sum_t$ des consommations individuelles $price_t$. Le meilleur endroit pour effectuer le calcul correspondant est également le composant C_U puisqu'il a déjà accès aux arguments nécessaires à cette fin.

Les choix de localisation pour les trois fonctions S, F et $\sum_t$ sont rappelés dans le Tableau 6.3.

Sélection d'affaiblissements. L'application retenue ici est une application de facturation. La règlementation exige que cette donnée soit précisément établie car la facture doit refléter exactement la consommation de l'usager. Nous supposons donc qu'aucune technique d'affaiblissement n'est utilisée.

3. Nous conservons les dénominations M, U et P dans l'outil par souci de lisibilité.

Calcul	Localisation
$fee = \sum_t (price_t)$	C_U
$price_t = F(cons_t)$	C_U
$cons_t = S(ECONS_t)$	C_M

TABLEAU 6.3 – Localisation des calculs pour Ω_1.

Choix des types de confiance. Il est maintenant possible de choisir un type de confiance pour chacune des opérations S, F et $\sum_t$ de l'architecture. Ce choix doit s'effectuer pour les exigences d'intégrité de chaque type d'acteur concerné : l'usager et le fournisseur. Les compteurs n'ont pas d'exigence d'intégrité sur les données et ne sont donc plus mentionnés dans le reste de cette étape.

La confiance par attestation est le type de confiance choisi classiquement lorsqu'un compteur est chargé d'effectuer une mesure, le compteur étant supposé être un composant métrologique sécurisé et certifié.

Nous considérons d'abord les exigences d'intégrité du fournisseur sur le résultat final *fee* du calcul. Le fournisseur est assuré de la bonne mesure S par une confiance par attestation de la part du compteur. La confiance par le calcul est exclue pour ce qui est de l'opération F car celle-ci est localisée chez un composant contrôlé par l'usager U. Le concepteur décide plutôt de retenir une confiance par attestation, prévoyant par exemple que le composant C_U est un module de plateforme de confiance. Pour l'opération $\sum_t$, la confiance par attestation est aussi sélectionnée de la même façon que pour l'opération F. Le fournisseur doit donc avoir confiance dans les attestations de tous les autres composants de l'architecture et être capable de vérifier leur authenticité.

Pour ce qui est de l'usager U, celui-ci est assuré de l'intégrité de la mesure S par le même moyen que le fournisseur : il établit une confiance par attestation dans le composant C_M. En revanche, pour ce qui est de la confiance en l'intégrité des calculs de F et de $\sum_t$, la confiance par le calcul est retenue puisqu'il a été choisi d'y localiser ces calculs.

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 6.4.

Calcul	Confiance de U	Confiance de P
$fee = \sum_t (price_t)$	Calcul	Attestation
$price_t = F(cons_t)$	Calcul	Attestation
$cons_t = S(ECONS_t)$	Attestation	Attestation

TABLEAU 6.4 – Types de confiance pour Ω_1.

6.2.2.3 Finalisation

La finalisation de l'architecture consiste à la munir de canaux de communication. Ceux-ci doivent permettre de satisfaire les exigences de confidentialité et les contraintes de cohérence.

Ces étapes d'ajout de canaux de communication de variables et de déclarations constituent les deux étapes de cette phase telles que décrites dans la Section 3.5.3.

Communication des variables. Un canal de communication doit être ajouté entre le composant du compteur C_M et le composant de l'usager C_U pour les variables $cons_t$ (afin que le calcul de F puisse s'effectuer). Un autre canal doit être rajouté de C_U vers le composant du fournisseur C_P pour qu'il puisse avoir connaissance de la variable *fee*.

Communication des déclarations. Le compteur envoie une attestation de l'intégrité de la mesure $cons_t = S(ECONS_t)$ à l'usager qui la fait suivre au fournisseur. L'usager envoie lui-aussi une attestation au fournisseur pour le reste des calculs des consommations finales $price_t = F(cons_t)$ et $fee = \sum_t (price_t)$.

L'ajout de ces communications clôt la phase de conception architecturale. La Figure 6.4 montre une capture d'écran de l'outil *CAPRIV* à cette étape.

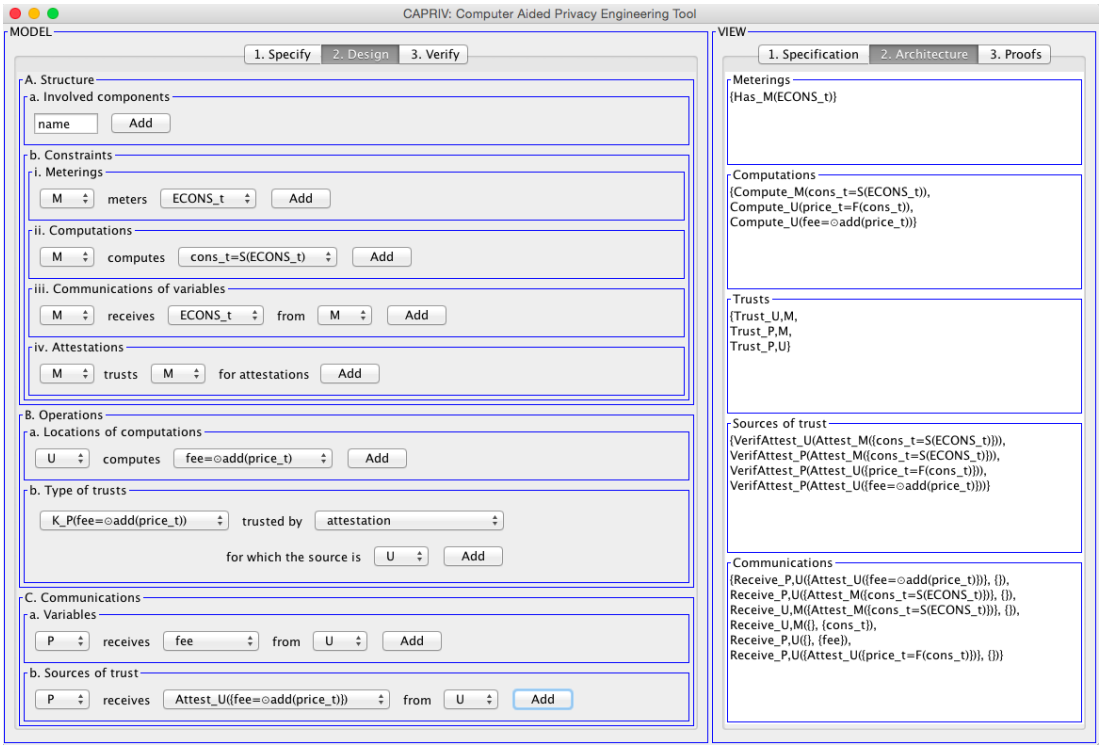

Figure 6.4 – Phase de conception dans *CAPRIV* pour Ω_1.

La Figure 6.5 représente l'architecture exprimée à l'aide de primitives formelles [4]. Le compteur reçoit la variable d'entrée $ECONS_t$ et effectue la mesure par l'application de S. Cette mesure $cons_t$ est envoyée au composant de l'usager pour effectuer les calculs de tarification F et

4. Comme dans le Chapitre 4, les flèches n'ont pas de sémantique et servent seulement à faciliter la lecture du schéma.

d'agrégation $\odot+$ pour déterminer le montant *fee* de la facture. Celui-ci est ensuite envoyé au composant du fournisseur avec une attestation sur la mesure S par C_{M} et une attestation sur la tarification F et l'agrégation $\odot+$ par le composant de l'usager C_{U}. La vérification de l'authenticité de ces attestations permet au composant du fournisseur, et enfin au fournisseur lui-même de recevoir le montant *fee* en étant convaincu de son intégrité (relativement aux $price_t$, $cons_t$ et $ECONS_t$).

L'existence d'un modèle formel sous-jacent permet d'effectuer une vérification formelle de l'architecture. Cette phase est détaillée dans la section suivante.

6.2.3 Vérification

La vérification de l'architecture permet au concepteur d'obtenir des garanties fortes sur les propriétés de cette architecture.

Cette phase se scinde en trois étapes comme détaillé dans la Section 3.6 : la vérification de la cohérence de l'architecture dans la Section 6.2.3.1, la vérification de la satisfaction des exigences dans la Section 6.2.3.2 et la vérification de la satisfaction des contraintes dans la Section 6.2.3.3.

6.2.3.1 Cohérence de l'architecture

L'architecture conçue est cohérente au regard des contraintes de cohérence (voir Section 4.2.3). Les canaux de communication de variables permettent aux calculs d'être effectués aux localisations choisies (telles que décrites dans le Tableau 6.3) et chaque variable n'est ainsi calculée qu'une seule fois.

6.2.3.2 Satisfaction des exigences

Pour vérifier la satisfaction des exigences, le concepteur peut se reposer sur l'outil *CAPRIV* qui offre une vérification formelle. *CAPRIV* indique si la propriété considérée est satisfaite. Si c'est le cas, il indique les règles de l'axiomatique de la Section 4.4.2 utilisées pour la prouver.

Le résultat de la vérification des exigences de confidentialité est présenté dans le Tableau 6.5 [5]. La vérification des propriétés de confidentialité Has^{all} (accès autorisé) s'appuie sur les règles correspondant aux opérations de communication (H2), de calcul (H3) et de dépendance à travers la relation *Dep* (H5). La vérification des propriétés de confidentialité Has^{none} (accès interdit) s'appuie uniquement sur la règle (H7). Cette règle repose sur l'absence de primitives permettant d'obtenir un quelconque accès aux variables considérées. Les accès autorisés sont garantis par la localisation des calculs et les canaux de communication ajoutés lors de l'étape de finalisation. En ce qui concerne les interdictions d'accès, le fournisseur P ne peut déduire des *fee* les détails des consommations car la fonction $\sum_t$ n'est pas inversible en raison de son caractère agrégatif.

Une capture d'écran de *CAPRIV* pour la vérification de la propriété Has^{none}_{P} ($ECONS_t$) (accès interdit à $ECONS_t$ pour le fournisseur) est donnée en Figure 6.6.

5. Comme dans le Chapitre 4, le premier élément de chaque paire représente l'entité concernée par l'exigence et le second élément l'ensemble des règles justifiant sa satisfaction.

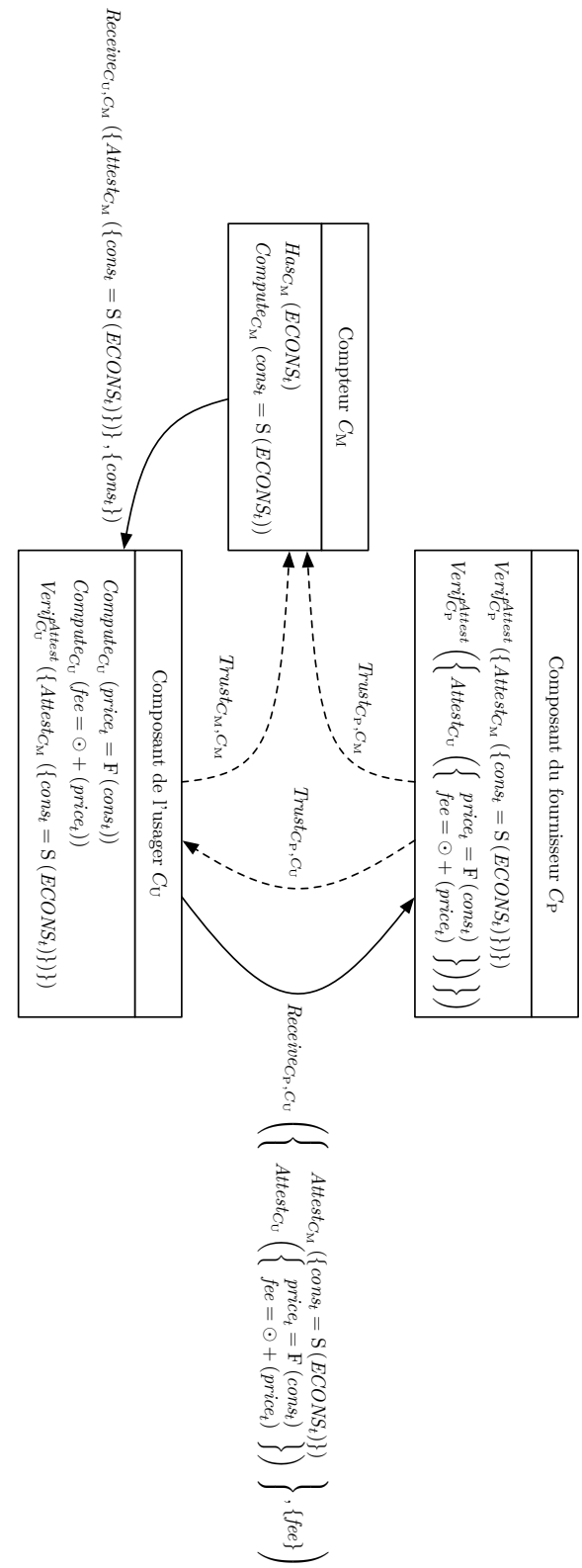

Figure 6.5 – Représentation dans le langage $\mathcal{L}_{FPA}$ de l'architecture pour Ω_1.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	(C_U, [H2, H5])	(C_P, [H7])
$cons_t$	(C_U, [H2])	(C_P, [H7])
$price_t$	(C_U, [H3])	(C_P, [H7])
fee	(C_U, [H3]), (C_P, [H2])	

TABLEAU 6.5 – Vérification des exigences de confidentialité pour Ω_1.

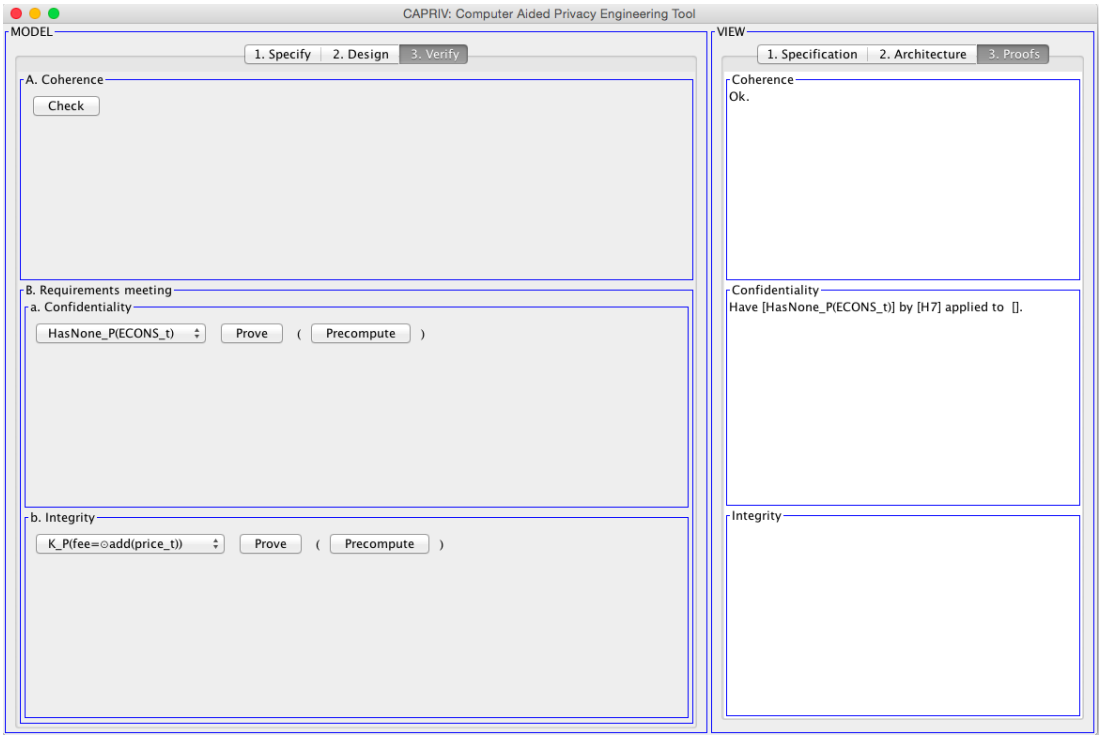

FIGURE 6.6 – Vérification de Has_P^{none} ($ECONS_t$) dans *CAPRIV* pour Ω_1.

Le Tableau 6.6 montre de la même manière la satisfaction des propriétés d'intégrité. Les règles utilisées sont (K1) pour la confiance par le calcul et (K5) pour la confiance par attestation. Les types de confiance sélectionnés permettent ainsi aux exigences d'intégrité d'être satisfaites.

Relations	Connaissance
$fee = \sum_t (price_t)$	(C_U, [K1]), (C_P, [K5])
$price_t = F(cons_t)$	(C_U, [K1]), (C_P, [K5])
$cons_t = S(ECONS_t)$	(C_U, [K5]), (C_P, [K5])

TABLEAU 6.6 – Vérification des exigences d'intégrité pour Ω_1.

Une capture d'écran de Why3 avec une théorie générée par *CAPRIV* pour la vérification de la propriété $K_P(fee = \odot + price_t)$ est donnée en Figure 6.7. Les axiomes de cette théorie

correspondent aux relations dérivées sans utilisation de (K▷).

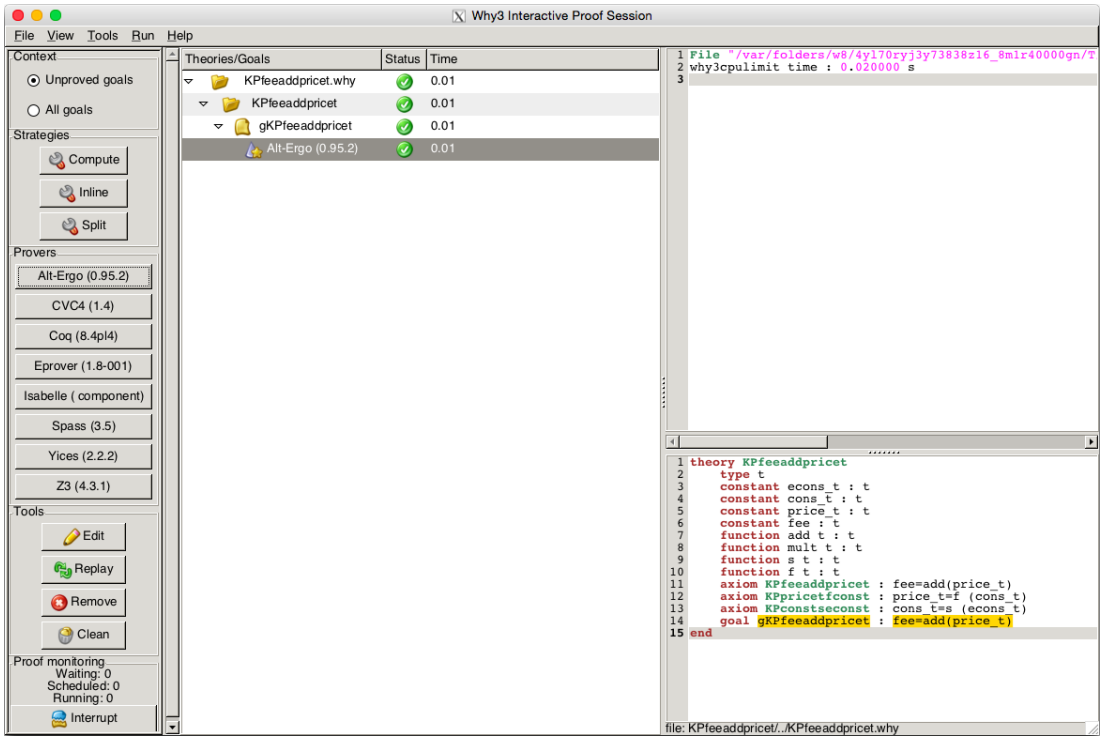

Figure 6.7 – Vérification de K_P ($fee = \odot + price_t$) dans *Why3* pour Ω_1.

Cette dernière étape permet au concepteur de s'assurer que les exigences définies sont bien satisfaites. Si certaines ne l'étaient pas, il lui faudrait alors revenir dans les étapes de spécification ou de conception (voir Sections 3.4 et 3.5 respectivement).

La satisfaction des exigences est une propriété indispensable qui doit être atteinte par une architecture. Cette dernière doit aussi satisfaire les contraintes non fonctionnelles comme illustré dans la section qui suit.

6.2.3.3 Satisfaction des contraintes

La solution architecturale obtenue par le concepteur doit s'analyser dans son contexte d'application qui est la facturation de la consommation d'électricité basée sur des compteurs intelligents. Ces compteurs ne sont pas mobiles et un lien de communication fiable peut donc être établi avec ceux-ci. De plus, leurs capacités de calcul sont réduites au minimum puisqu'ils ne sont chargés que des mesures. Les calculs sont ainsi déportés dans une unité sous le contrôle de l'utilisateur. Le fournisseur doit donc avoir confiance dans les attestations de cette unité.

Cependant, cette architecture n'est pas commune. Elle pose donc la question de son acceptabilité par les usagers en fonction de l'ergonomie du système final et des fournisseurs qui doivent accepter de perdre la main sur une partie de l'architecture qui leur était acquise jusqu'ici et de faire confiance à de nouvelles unités logiques.

Ces limitations invitent à tenter de créer une deuxième architecture. Cette démarche est proposée dans la section suivante.

6.3 Deuxième solution : confiance par sécurité

Une deuxième solution architecturale est proposée afin de pallier les limitations de la première. Cette nouvelle solution est basée sur la confiance par sécurité. Au lieu d'envoyer une simple attestation au fournisseur P, l'usager U va cette fois lui apporter une preuve. La Figure 6.8 représente de manière informelle l'architecture envisagée pour cette solution [6].

Nous ne décrivons pas dans le reste de ce chapitre l'ensemble de la démarche de manière aussi systématique que pour la première solution. Nous ne mentionnons ainsi que les différences et évolutions par rapport à cette dernière. L'itération sur la phase de spécification est décrite dans la Section 6.3.1, la phase de conception en Section 6.3.2 et la phase de vérification en Section 6.3.3.

6.3.1 Spécification

À ce stade, le concepteur de l'architecture souhaite revenir sur des choix faits précédemment. Il préfère adopter une confiance par sécurité pour les calculs effectués par le composant de l'usager (l'application de la tarification F et l'agrégation $\sum_t$). La représentation informelle de la Figure 6.8 montre que les deux relations seront prouvées au fournisseur P par l'usager U. Nous considérons dans la suite que la preuve de $price_t = \mathrm{F}(cons_t)$ est une primitive de base et nous détaillons la preuve de $fee = \sum_t (price_t)$.

La construction de cette deuxième solution nécessite une extension du modèle de service comme décrit en Section 6.3.1.1. Par suite, les exigences sont redéfinies dans la Section 6.3.1.2.

6.3.1.1 Extension du modèle de service

Il est nécessaire d'étendre le modèle pour détailler la preuve de la relation $fee = \sum_t (price_t)$. Nous allons exploiter le fait qu'il s'agisse d'une agrégation basée sur l'addition et introduisons la fonction HH. Celle-ci servira au fournisseur à effectuer des calculs sur des valeurs modifiées des variables. Ces valeurs modifiées conservent néanmoins une structure homomorphe aux valeurs non modifiées. Le fournisseur ne devra pas pouvoir retrouver les valeurs non modifiées à partir des valeurs modifiées. Cette fonction ne doit donc pas être inversible et doit être injective. HH doit en plus admettre une propriété d'homomorphisme telle que $\mathrm{HH}(x+y) = \mathrm{HH}(x) \times \mathrm{HH}(y)$. La fonction HH correspond à une fonction de hachage. Le modèle du service Ω_1 est donc étendu

6. Comme dans la section précédente, nous donnons cette représentation en amont de la démarche par souci de clarté. Le concepteur n'a pas besoin *a priori* de cette représentation pour suivre la démarche systématique.

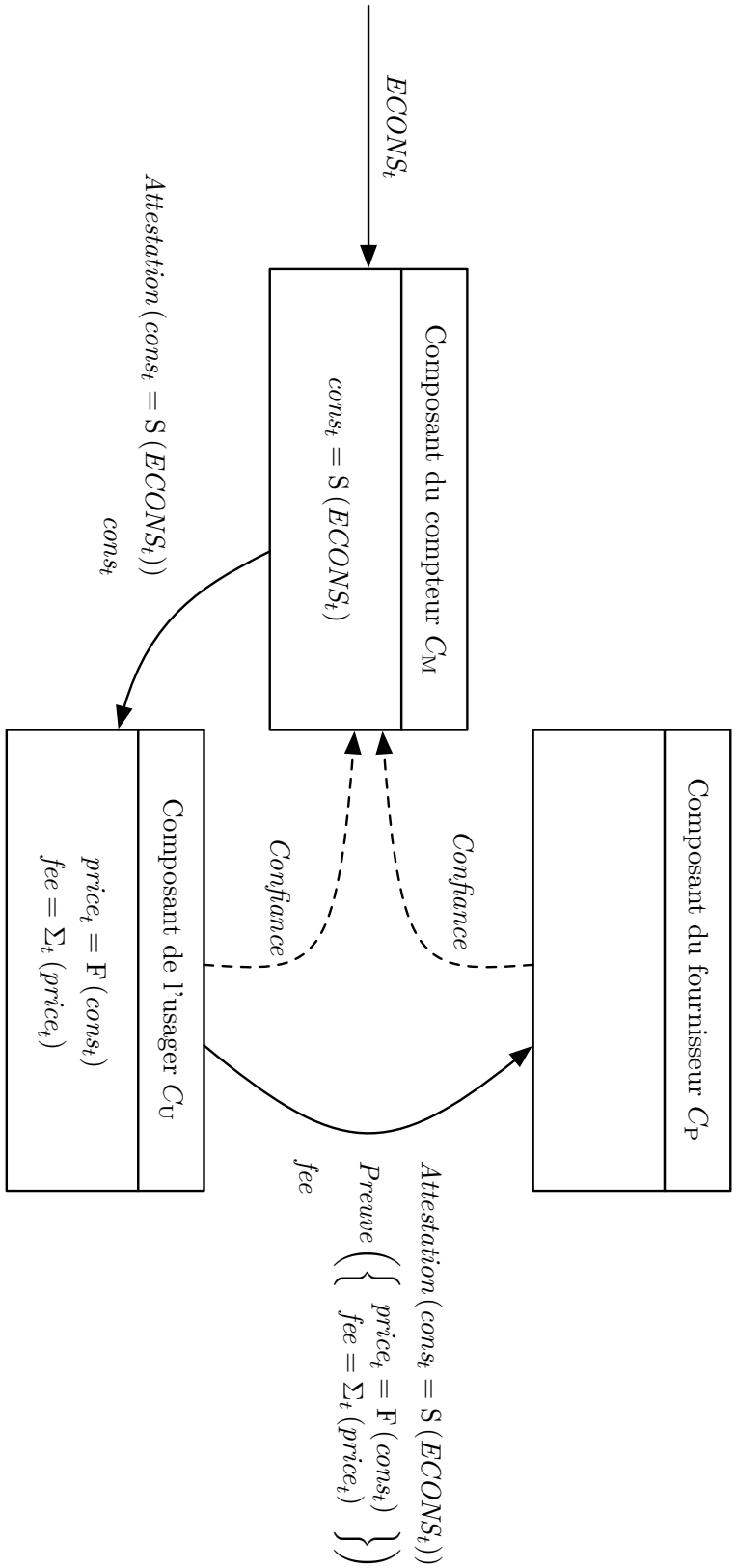

Figure 6.8 – Représentation informelle de l'architecture pour Ω_2.

pour devenir :

$$\Omega_2 = \left\{ \begin{array}{c} fee = \sum\limits_t (price_t) \\ price_t = \mathrm{F}\,(cons_t) \\ cons_t = \mathrm{S}\,(ECONS_t) \\ cfee = \mathrm{HH}\,(fee) \\ cfee = \prod\limits_t (cprice_t) \\ cprice_t = \mathrm{HH}\,(price_t) \end{array} \right\}$$

avec les variables *cfee* et $cprice_t$ les hachés des variables *fee* et $price_t$. La version graphique de ce service est représentée en Figure 6.9. Cette *PET* correspond au chiffrement homomorphe introduit dans la Section 2.1.3.

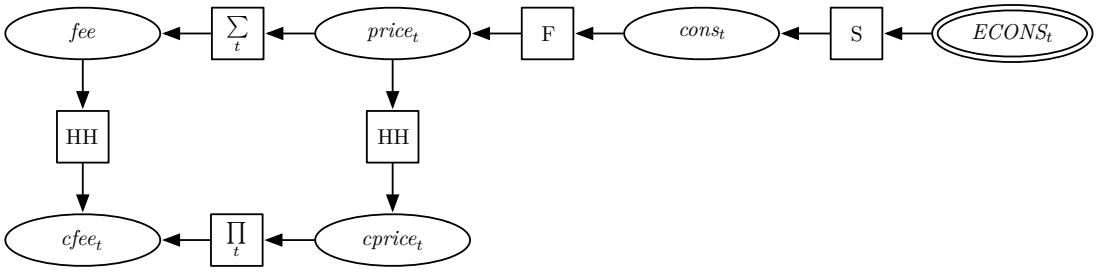

FIGURE 6.9 – Modèle du service Ω_2.

6.3.1.2 Redéfinition des exigences

Les nouvelles variables et équations apparues dans Ω_2 nécessitent de redéfinir les exigences. Les exigences n'ont cependant pas changé pour les variables et équations déjà présentes dans Ω_1. Les nouvelles variables introduites qui sont le résultat de fonction de hachage (non inversible) peuvent être communiquées au fournisseur comme montré dans le Tableau 6.7.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	U	P
$cons_t$	U	P
$price_t$	U	P
fee	U, P	
$cprice_t$	U, P	
cfee	U, P	

TABLEAU 6.7 – Spécification des exigences de confidentialité pour Ω_2.

Les exigences de confidentialité déjà présentes dans le cas de Ω_1 ne changent pas non plus et seul le fournisseur est intéressé par l'intégrité des nouvelles équations (la modification des types de confiance pour l'usager n'est pas envisagée pour cette nouvelle solution). Le Tableau 6.8 montre la spécification des exigences d'intégrité qui en résulte.

Relations	Connaissance
$fee = \sum_t (price_t)$	U, P
$price_t = \mathrm{F}(cons_t)$	U, P
$cons_t = \mathrm{S}(ECONS_t)$	U, P
$cfee = \mathrm{HH}(fee)$	P
$cfee = \prod_t (cprice_t)$	P
$cprice_t = \mathrm{HH}(price_t)$	P

Tableau 6.8 – Spécifications des exigences d'intégrité pour Ω_2.

Nous montrons dans la Figure 6.10 la vue de *CAPRIV* avec le modèle enrichi par les nouvelles équations de Ω_2.

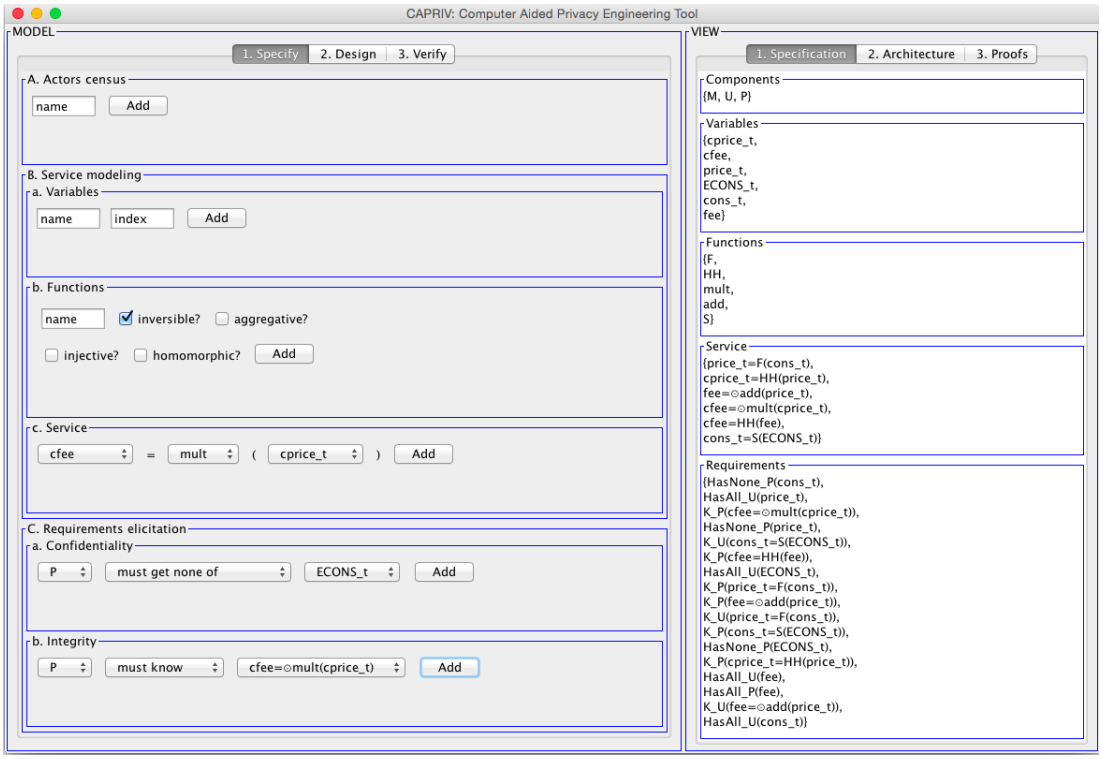

Figure 6.10 – Phase de spécification dans *CAPRIV* pour Ω_2.

La suite de la phase de spécification est la phase de conception qui est décrite dans la section suivante.

6.3.2 Conception architecturale

L'introduction de la fonction HH et des variables *cfee* et $cprice_t$ permet d'effectuer de nouveaux calculs dans l'architecture. Nous ne détaillons que les étapes de localisation des calculs et de

choix des types de confiance dans les paragraphes suivants. En effet, ces deux étapes sont les plus impactées par l'itération en cours.

La localisation des nouveaux calculs est détaillée dans la Section 6.3.2.1 et la manière dont les types de confiance sont choisis est détaillée dans la Section 6.3.2.2.

6.3.2.1 Localisation des calculs

La localisation des calculs pour les fonctions S, F et $\sum_t$ ne change pas par rapport à la première solution. Les nouvelles équations introduites dans le modèle sont considérées tour à tour.

La meilleure localisation pour le calcul de $cprice_t = \mathrm{HH}(price_t)$ est le composant de l'usager C_U : il est déjà chargé du calcul des $price_t$ et a donc les arguments nécessaires au calcul.

Le calcul de $cfee = \prod_t (cprice_t)$ pourrait aussi s'effectuer auprès du composant C_U puisqu'il dispose des arguments nécessaires pour ce faire. Cela nécessiterait cependant, dans l'étape suivante, de choisir un type de confiance approprié. Nous serions alors dans la même situation que pour la première solution (avec le calcul de $\prod_t$ au lieu de $\sum_t$). Le concepteur change donc de localisation et confie ce calcul au composant du fournisseur C_P (sans introduire de problème de confidentialité car la fonction HH n'est pas inversible : les $price_t$ ne peuvent donc être déduits des $cprice_t$).

Enfin, le calcul de l'équation $cfee = \mathrm{HH}(fee)$ n'est pas effectué. En effet, *cfee* est déjà calculé par ailleurs et il n'est pas possible de calculer deux fois la même variable (cela entraînerait une incohérence).

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 6.9.

Calcul	Localisation
$fee = \sum_t (price_t)$	C_U
$price_t = \mathrm{F}(cons_t)$	C_U
$cons_t = \mathrm{S}(ECONS_t)$	C_M
$cfee = \mathrm{HH}(fee)$	
$cfee = \prod_t (cprice_t)$	C_P
$cprice_t = \mathrm{HH}(price_t)$	C_U

TABLEAU 6.9 – Localisation des calculs pour Ω_2.

Après avoir localisé les calculs, nous choisissons les types de confiance appropriés comme détaillé dans la section suivante.

6.3.2.2 Choix des types de confiance

Les types de confiance appropriés sont choisis pour chaque calcul. Pour le composant U, les types de confiance ne changent pas par rapport à la première solution. U n'a pas besoin d'avoir confiance dans l'intégrité des trois nouveaux calculs puisque celles déjà existantes lui suffisent pour satisfaire l'exigence d'intégrité sur $fee = \sum_t (\mathrm{F}(\mathrm{S}(ECONS_t)))$.

Pour ce qui est du fournisseur P, les exigences d'intégrité portent également sur les trois équations permettant de déduire $fee = \sum_t (\mathrm{F}(\mathrm{S}(ECONS_t)))$. Nous avons cependant choisi de détailler la preuve de $fee = \sum_t (price_t)$ en introduisant les trois autres équations. Il devient dès lors nécessaire que P puisse vérifier l'intégrité des trois équations de calcul introduites.

Le type de confiance choisi pour $cprice_t = \mathrm{HH}(price_t)$ est la confiance par sécurité : l'usager C_U doit envoyer une preuve de cette relation à C_P. Les deux autres calculs $cfee = \prod_t (cprice_t)$ et $cfee = \mathrm{HH}(fee)$ sont associés à un type de confiance par calcul pour le fournisseur P. Ce choix est naturel pour le premier calcul qui est localisé sur ce même composant. Le fournisseur dispose des paramètres nécessaires pour établir la confiance dans le deuxième calcul en effectuant localement un test d'égalité entre *cfee* et HH(*fee*). La confiance dans ces trois équations permet d'établir la confiance dans $fee = \sum_t (price_t)$ pour laquelle nous retenons un type de confiance par sécurité.

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 6.10.

Calcul	Confiance de U	Confiance de P
$fee = \sum_t (price_t)$	Calcul	Sécurité
$price_t = \mathrm{F}(cons_t)$	Calcul	Sécurité
$cons_t = \mathrm{S}(ECONS_t)$	Attestation	Attestation
$cfee = \mathrm{HH}(fee)$		Calcul
$cfee = \prod_t (cprice_t)$		Calcul
$cprice_t = \mathrm{HH}(price_t)$		Sécurité

Tableau 6.10 – Types de confiance pour Ω_2.

L'interface de l'outil *CAPRIV* à l'issue de cette étape est montrée en Figure 6.11. Elle comprend les choix de localisation et de types de confiance. Les éléments obtenus à travers la phase de finalisation (non développés dans cette section) ont aussi été entrés.

Le modèle formel obtenu à l'issue de cette phase est représenté sur la Figure 6.12. Il se distingue notamment du modèle de la première solution (voir Figure 6.5) par le changement des types de confiance choisis et l'introduction de nouveaux calculs nécessaires à leur mise en œuvre.

L'architecture obtenue pour cette deuxième solution (après application de l'étape de finalisation, non développée ici) semble satisfaire les exigences. La phase de vérification développée dans la section suivante permet de le vérifier.

6.3.3 Vérification

Nous nous concentrons à ce stade sur l'étape de validation des exigences. Le résultat de la vérification des exigences de confidentialité est présenté dans le Tableau 6.11. Nous pouvons noter que la propriété de non-inversibilité de la fonction HH est nécessaire afin que le fournisseur, qui a accès aux $cprice_t$, ne puisse pas retrouver les $price_t$ (à travers la relation *Dep*).

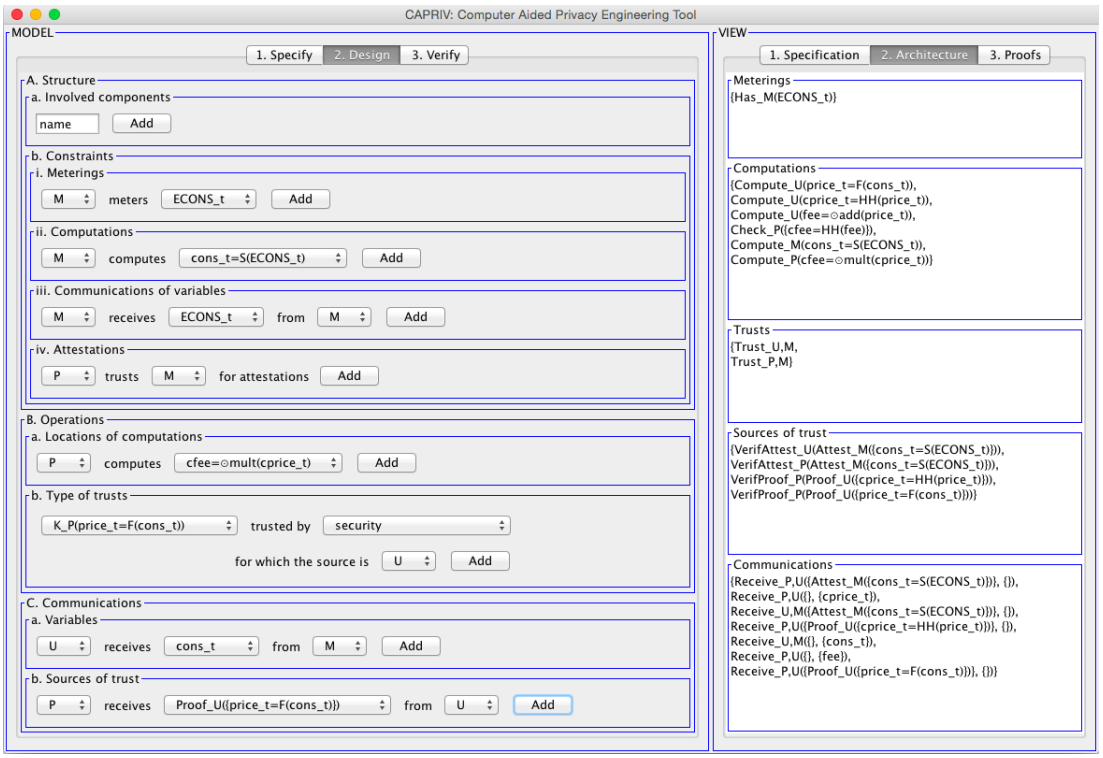

FIGURE 6.11 – Phase de conception dans *CAPRIV* pour Ω_2.

Variables	Accès autorisé	Accès interdit
$ECONS_t$	(C_U, [H2, H5])	(C_P, [H7])
$cons_t$	(C_U, [H2])	(C_P, [H7])
$price_t$	(C_U, [H3])	(C_P, [H7])
fee	(C_U, [H3]), (C_P, [H2])	
$cprice_t$	(C_U, [H3]), (C_P, [H2])	
$cfee$	(C_U, [H3, H5]), (C_P, [H3])	

TABLEAU 6.11 – Vérification des exigences de confidentialité pour Ω_2.

La Figure 6.13 montre une capture d'écran de l'outil *CAPRIV* pour la vérification de la propriété $Has^{all}_U(cons_t)$ (accès autorisé aux $cons_t$ pour l'usager). *CAPRIV* indique que la satisfaction de cette exigence est prouvée par l'application de la règle (H2) (car C_U reçoit directement $cons_t$ de C_M).

Le Tableau 6.12 montre de la même manière la satisfaction des propriétés d'intégrité. C_U a confiance dans les relations pour les mêmes raisons que dans la première solution. En revanche, les règles utilisées pour justifier de la confiance de P sont différentes. P a confiance dans la mesure S par l'application de la règle (K5) et dans la tarification F par application de la règle (K3). Pour l'agrégation, la règle (K$_\triangleright$) est utilisée : l'assurance de la connaissance de $fee = \sum_t (price_t)$ à partir des trois équations faisant intervenir HH et $\prod_t$ fait intervenir les

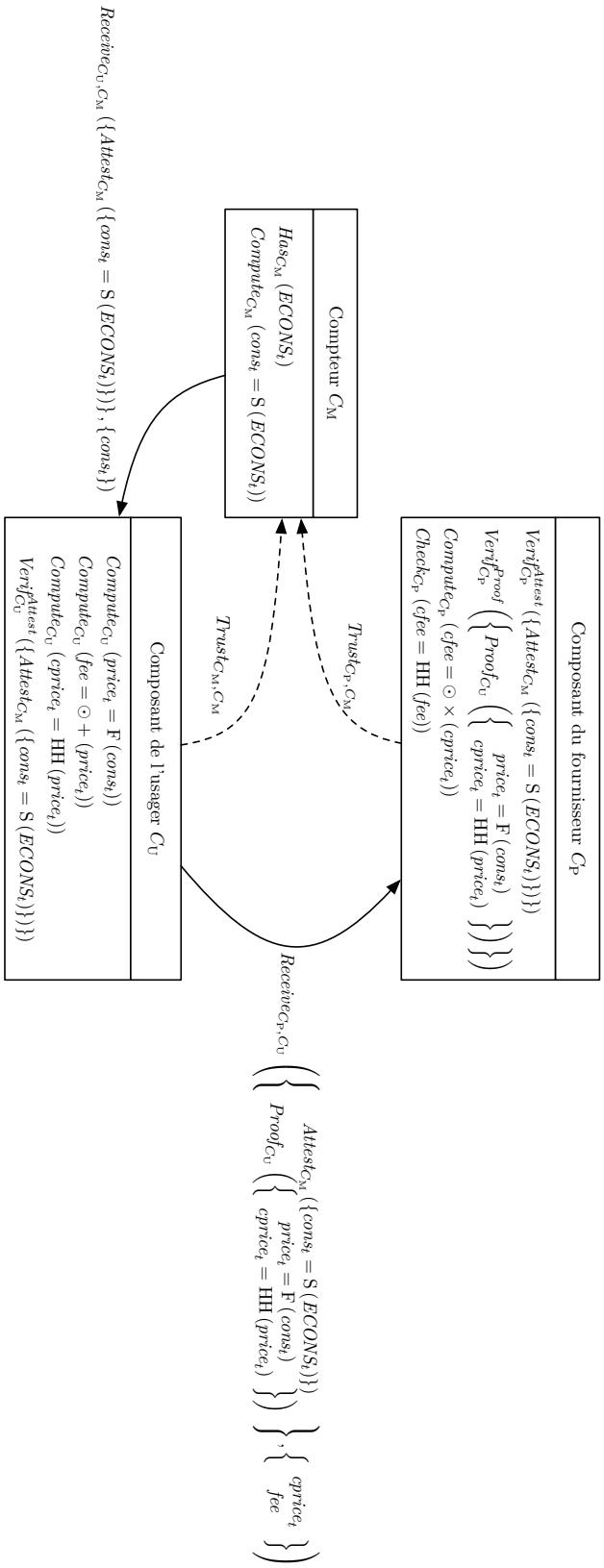

Figure 6.12 – Représentation dans le langage $\mathcal{L}_{FPA}$ de l'architecture pour Ω_2.

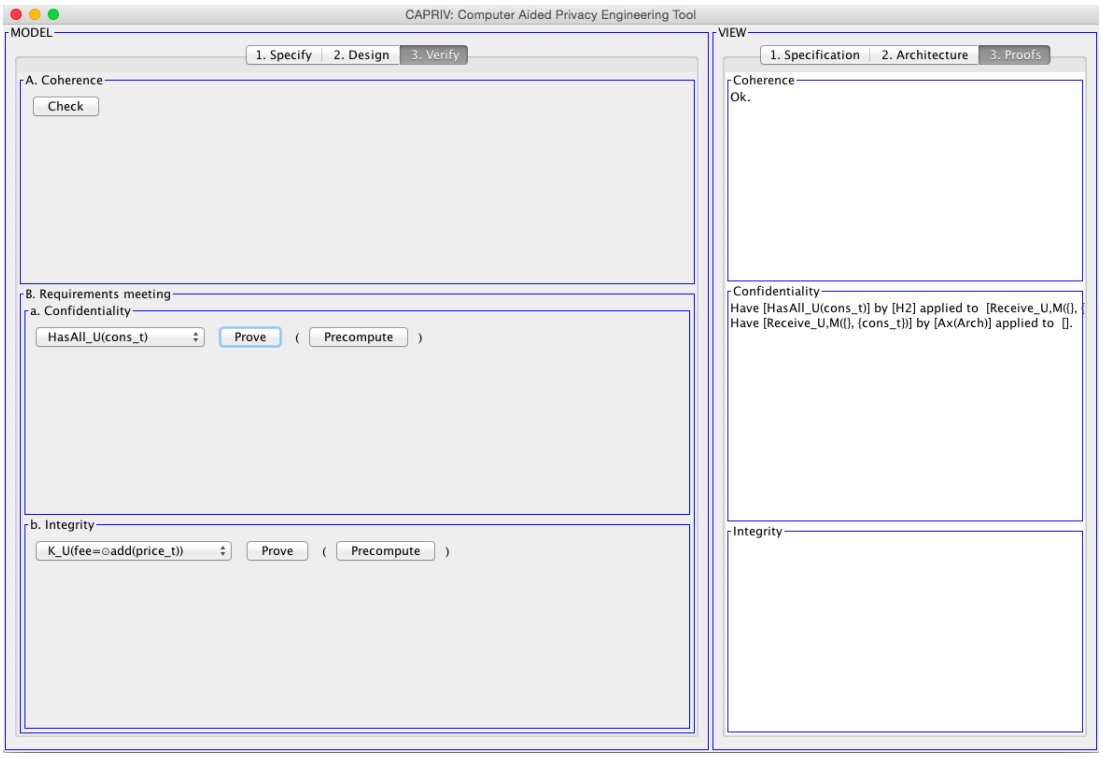

FIGURE 6.13 – Vérification de $Has^{all}_{U}(cons_t)$ dans *CAPRIV* pour Ω_2.

capacités de déduction $\triangleright_{C_P}$ du composant du fournisseur.

Relations	Connaissance
$fee = \sum_t (price_t)$	$(C_U, [K1]), (C_P, [K1, K2, K3, K\triangleright])$
$price_t = F(cons_t)$	$(C_U, [K1]), (C_P, [K3])$
$cons_t = S(ECONS_t)$	$(C_U, [K5]), (C_P, [K5])$
$cfee = HH(fee)$	$(C_P, [K2])$
$cfee = \prod_t (cprice_t)$	$(C_P, [K1])$
$cprice_t = HH(price_t)$	$(C_P, [K3])$

TABLEAU 6.12 – Vérification des exigences d'intégrité pour Ω_2.

Une capture d'écran de Why3 avec une théorie générée par *CAPRIV* pour la vérification de la propriété $K_P(fee = \odot + price_t)$ est donnée par la Figure 6.14. La théorie automatiquement générée par *CAPRIV* et transmise à Why3 comprend un axiome d'injectivité (`hh_inj`) et un axiome d'homomorphisme (`hh_hom`) pour la fonction HH. Les autres équations justifiées par (K1), (K2) et (K3) apparaissent aussi car elles sont utilisées dans la résolution équationnelle. Cette résolution modélise l'application de la règle (K$\triangleright$).

Cette deuxième solution satisfait les exigences de confidentialité et d'intégrité. Elle repose à la fois sur la confiance par attestation pour ce qui est de la mesure S et sur la confiance par sécurité

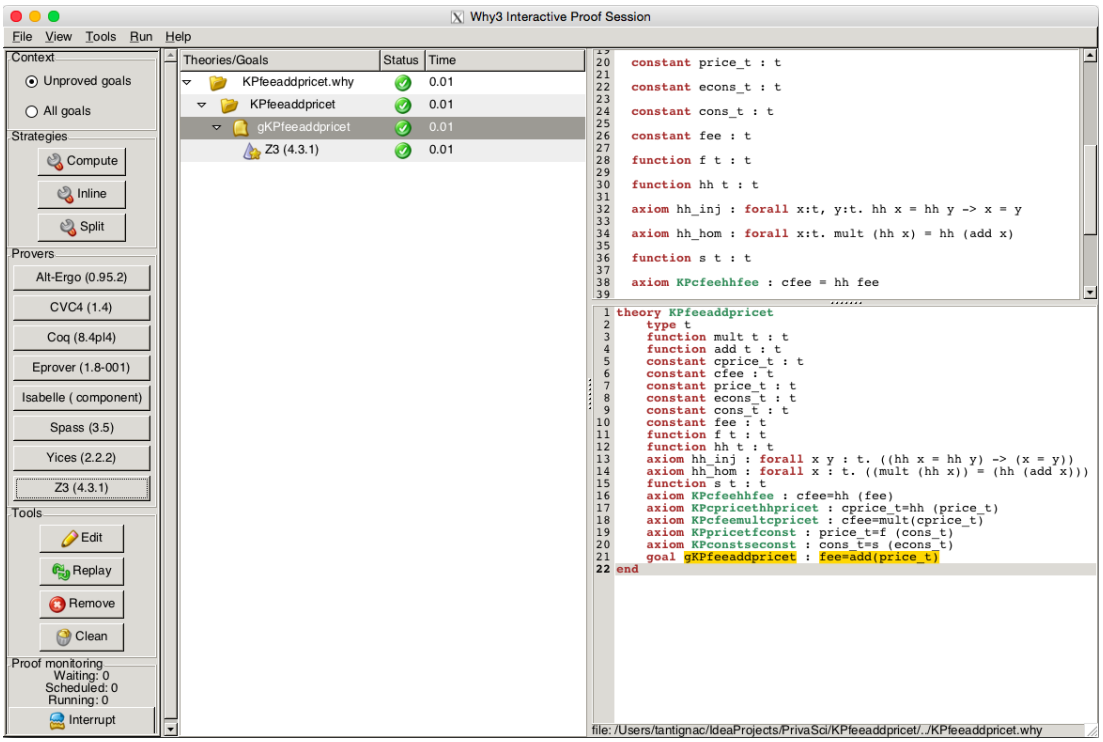

Figure 6.14 – Vérification de K_P ($fee = \odot + price_t$) dans *Why3* pour Ω_2.

pour la tarification F et l'agrégation $\sum_t$.

Cette solution, qui correspond à la proposition de [RD10], paraît aussi plus acceptable pour le fournisseur. En effet, il n'a plus besoin d'établir un lien de confiance par attestation en faveur de l'usager, ce qui paraît plus réaliste.

Il reste cependant encore un lien de confiance par attestation en faveur du compteur. Celui-ci est généralement une unité métrologique certifiée par un tiers de confiance. Le fournisseur peut désirer avoir la possibilité de vérifier que les compteurs fonctionnent correctement. Nous proposons une troisième solution faisant intervenir la confiance par redevabilité pour satisfaire cette exigence.

6.4 Troisième solution : confiance par redevabilité

La troisième solution architecturale à concevoir doit permettre au fournisseur P de vérifier le bon fonctionnement du compteur. Pour ce faire, la solution sera basée sur la confiance par redevabilité. Le fournisseur prélève un échantillon des $ECONS_t$ à la source pour vérifier que la fonction S a bien été appliquée. La Figure 6.15 représente de manière informelle l'architecture envisagée pour cette solution. Le lien de confiance de C_P vers C_M a été enlevé et un échantillonnage a été ajouté par rapport au schéma informel de la solution précédente (voir Figure 6.8). La preuve de l'agrégation (détaillée dans la section précédente) apparaît maintenant de manière

simplifiée comme une primitive de base au même titre que la preuve concernant la tarification.

Nous présentons la nouvelle itération pour la phase de spécification dans la Section 6.4.1, pour la phase de conception dans la Section 6.4.2 et pour la phase de vérification dans la Section 6.4.3.

6.4.1 Spécification

Comme pour la deuxième solution, le changement de type de confiance nécessite une nouvelle extension du modèle de service. Une redéfinition des exigences sera nécessaire à la fois en raison de l'évolution du modèle de service et du changement du type de confiance.

L'étape d'extension du modèle de service est présentée dans la Section 6.4.1.1 et la redéfinition des exigences dans la Section 6.4.1.2.

6.4.1.1 Extension du modèle de service

Afin de permettre au fournisseur de prélever un échantillon et de s'en servir, le modèle de service doit être étendu autour de la relation $cons_t = \mathrm{S}(ECONS_t)$. Nous allons utiliser un schéma d'engagement (comme introduit dans la Section 2.1.3). La fonction de hachage H est introduite à cette fin. Celle-ci doit être injective et non inversible. Elle servira à engager les valeurs des consommations fournies par le compteur (sous leur forme hachée) et à vérifier ultérieurement leur conformité avec l'échantillon prélevé. Le modèle du service Ω_1 est donc étendu et devient :

$$\Omega_3 = \left\{ \begin{array}{c} fee = \sum_t (price_t) \\ price_t = \mathrm{F}(cons_t) \\ cons_t = \mathrm{S}(ECONS_t) \\ cons'_t = \mathrm{S}(ECONS_t) \\ ccons_t = \mathrm{H}(cons_t) \\ ccons'_t = \mathrm{H}(cons'_t) \\ ccons'_t = \mathrm{Id}(ccons_t) \end{array} \right\}$$

avec les variables $ccons_t$ les hachés des variables $cons_t$ et les variables $cons'_t$ et $ccons'_t$ de nouvelles variables pour les mesures et leurs hachés respectivement. La fonction Id est introduite pour représenter la fonction identité. La version graphique de ce service est représentée en Figure 6.16.

L'étape suivante est la redéfinition des exigences en tenant compte des modifications apportées au modèle.

6.4.1.2 Redéfinition des exigences

L'utilisation de la confiance par redevabilité nécessite d'avoir des exigences moins fortes. En effet, au moins une valeur de $ECONS_t$ doit être divulguée dans le cadre du schéma d'engagement. Le niveau d'assurance obtenu est aussi plus faible puisque la vérification d'une valeur ne signifie pas que toutes sont correctes. Ces deux cas correspondent aux propriétés Has_i^{one} et B_i de $\mathcal{L}_{FPP}$ (voir Section 4.4.1) que nous qualifions ici d'« accès interdit (sauf un) » et de « croyance » respectivement.

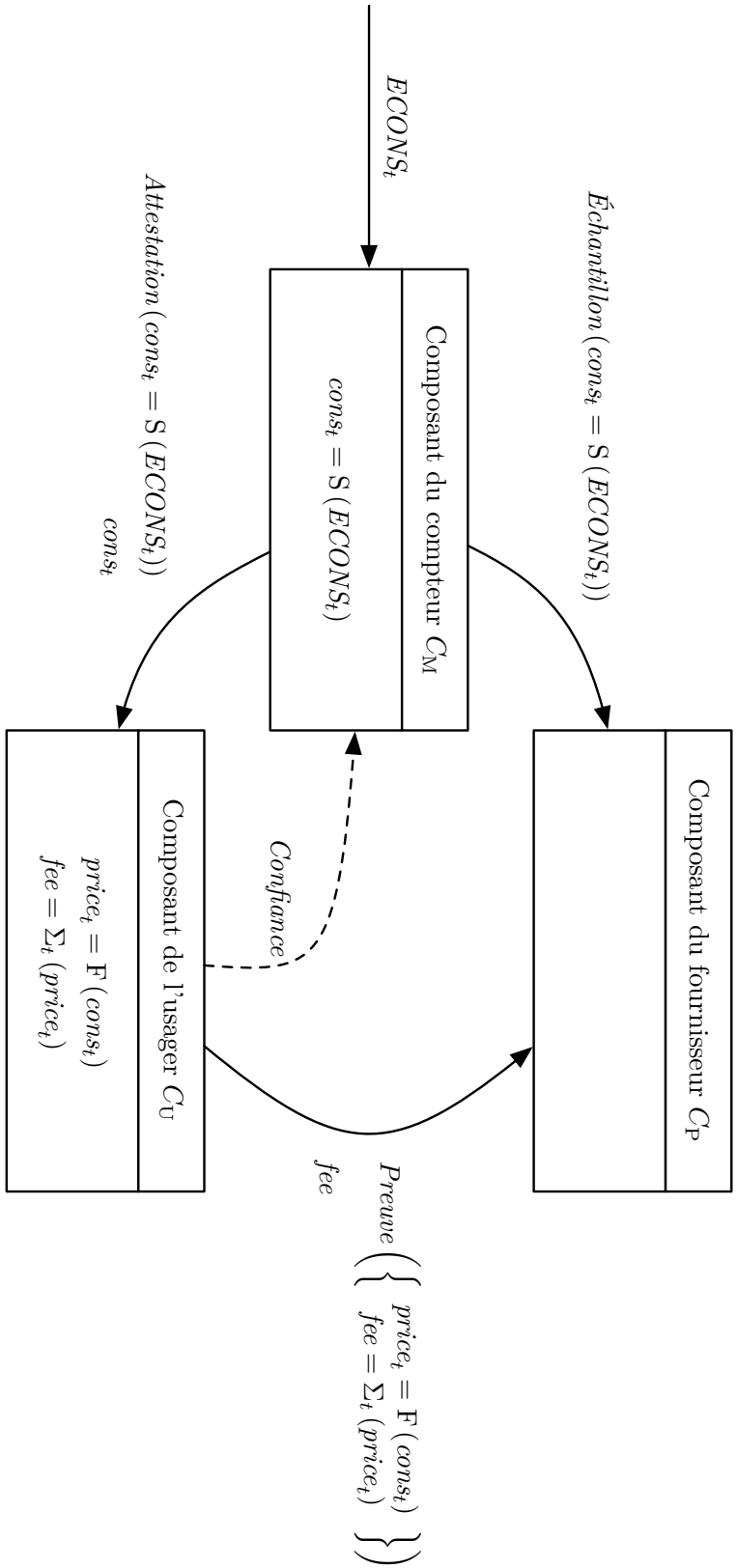

Figure 6.15 – Représentation informelle de l'architecture pour Ω_3.

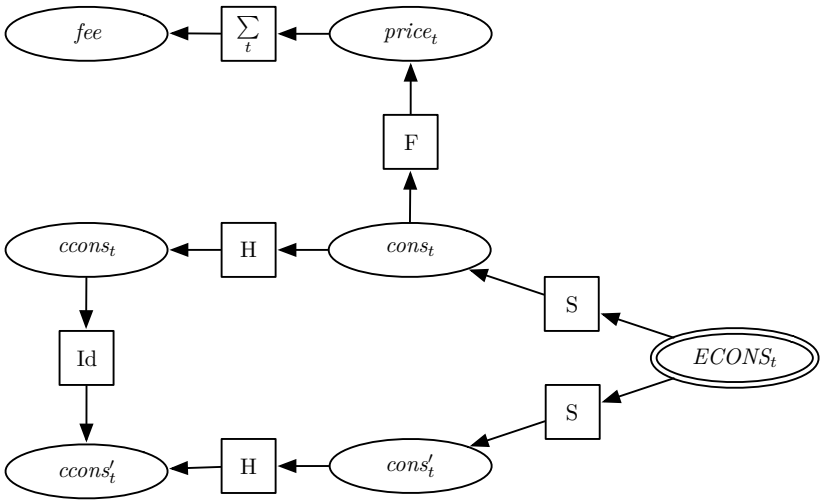

FIGURE 6.16 – Modèle du service Ω_3.

Les exigences de confidentialité et d'intégrité pour l'usager restent les mêmes que précédemment mais celles pour le fournisseur doivent être relaxées comme illustré dans le Tableau 6.13 pour les exigences de confidentialité et le Tableau 6.14 pour les exigences d'intégrité. Ces tableaux intègrent les nouvelles variables et équations du modèle de service Ω_3.

Variables	Accès autorisé	Accès interdit (sauf un)
$ECONS_t$	U	P
$cons_t$	U	P
$price_t$	U	P
fee	U, P	
$cons'_t$	U	P
$ccons_t$	U, P	
$ccons'_t$	U, P	

TABLEAU 6.13 – Spécification des exigences de confidentialité pour Ω_3.

Relations	Connaissance	Croyance
$fee = \sum_t (price_t)$	U	P
$price_t = \mathrm{F}(cons_t)$	U	P
$cons_t = \mathrm{S}(ECONS_t)$	U	P
$cons'_t = \mathrm{S}(ECONS_t)$		P
$ccons_t = \mathrm{H}(cons_t)$		P
$ccons'_t = \mathrm{H}(cons'_t)$		P
$ccons'_t = \mathrm{Id}(ccons_t)$		P

TABLEAU 6.14 – Spécifications des exigences d'intégrité pour Ω_3.

La Figure 6.17 représente la vue de *CAPRIV* avec le modèle étendu Ω_3.

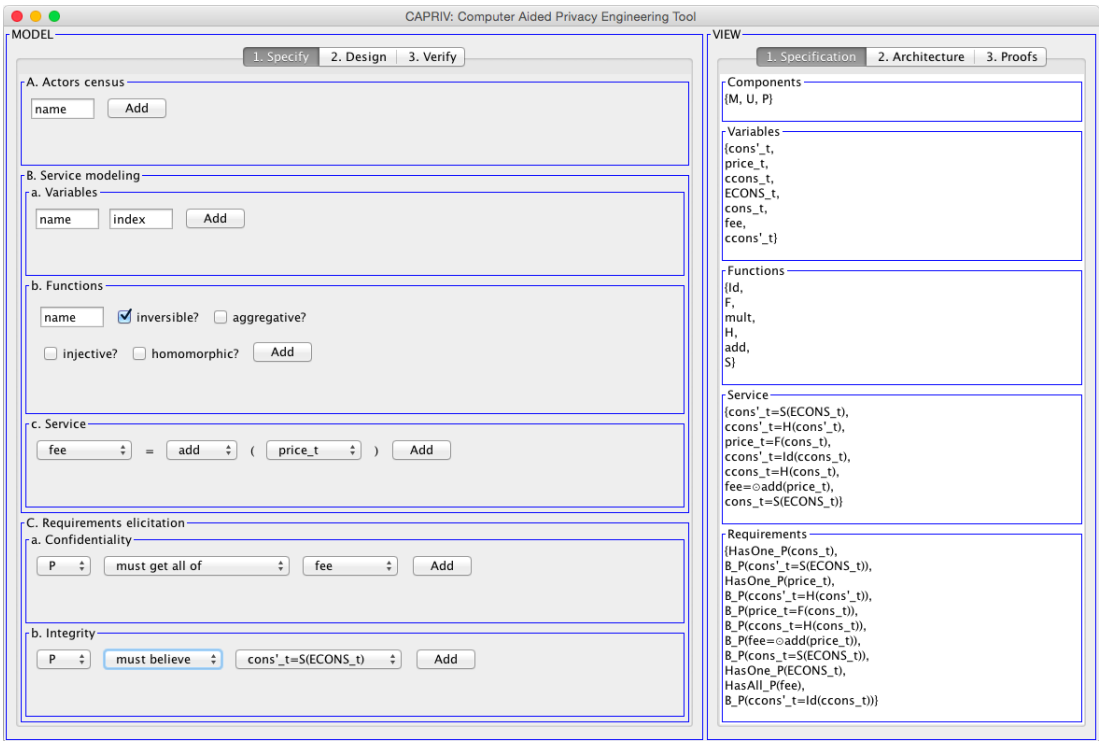

Figure 6.17 – Phase de spécification dans *CAPRIV* pour Ω_3.

Une fois la phase de spécification terminée, nous pouvons passer à la phase de conception qui est détaillée dans la section suivante.

6.4.2 Conception architecturale

Comme pour la deuxième solution, l'introduction des fonctions H et Id et des variables $ccons_t$, $cons'_t$ et $ccons'_t$ autorisent de nouveaux calculs dans l'architecture. Ces nouveaux calculs doivent être localisés et des types de confiance doivent être choisis quand nécessaire.

Nous détaillons l'étape de localisation des calculs dans la Section 6.4.2.1 et l'étape de choix des types de confiance dans la Section 6.4.2.2.

6.4.2.1 Localisation des calculs

La localisation des calculs pour les fonctions S, F et $\sum_t$ ne change pas par rapport à la première solution. Les nouvelles équations sont considérées tour à tour.

La meilleure localisation pour le calcul de $ccons_t = \mathrm{H}(cons_t)$ est le composant de l'usager C_U car celui-ci possède déjà les arguments utiles à ce calcul.

En revanche, il est plus facile (sans que cela constitue une obligation) de localiser le calcul des équations $cons'_t = \mathrm{S}(ECONS_t)$ et $ccons'_t = \mathrm{H}(cons'_t)$ auprès du composant chargé de vérifier

les valeurs engagées du schéma d'engagement. Il s'agit ici du composant C_P. Ces calculs ne seront cependant que partiels et porteront seulement sur les valeurs échantillonnées [7].

Enfin, l'équation $ccons'_t = \mathrm{Id}(ccons_t)$ ne donne pas lieu à un calcul dans l'architecture ($ccons_t$ et $ccons'_t$ sont déjà résultats d'autres calculs) pour respecter les contraintes de cohérence.

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 6.15.

Calcul	Localisation
$fee = \sum_t (price_t)$	C_U
$price_t = \mathrm{F}(cons_t)$	C_U
$cons_t = \mathrm{S}(ECONS_t)$	C_M
$cons'_t = \mathrm{S}(ECONS_t)$	C_P*
$ccons_t = \mathrm{H}(cons_t)$	C_U
$ccons'_t = \mathrm{H}(cons'_t)$	C_P*
$ccons'_t = \mathrm{Id}(ccons_t)$	

TABLEAU 6.15 – Localisation des calculs pour Ω_3 (* pour les calculs sur un échantillon).

Nous pouvons désormais choisir les types de confiance les plus appropriés pour les relations du modèle Ω_3.

6.4.2.2 Choix des types de confiance

De la même manière que pour le choix des localisations, les types de confiance sont choisis tour à tour pour chaque équation.

Pour le composant U, les types de confiance ne changent pas par rapport aux deux premières solutions : U reçoit la valeur mesurée de la consommation, a confiance dans les attestations du compteur et procède lui-même au calcul du montant à facturer.

Pour ce qui est du fournisseur P, les exigences d'intégrité portent sur les trois équations permettant de déduire $fee = \sum_t (\mathrm{F}(\mathrm{S}(ECONS_t)))$. Nous avons cependant choisi de détailler la confiance par redevabilité avec un schéma d'engagement et un relevé d'échantillon pour vérifier que $cons_t = \mathrm{S}(ECONS_t)$. Quatre nouvelles équations ont été introduites pour ce faire. Il devient dès lors nécessaire que P puisse vérifier l'intégrité de ces quatre équations.

Le type de confiance retenu pour les équations $fee = \sum_t (price_t)$, $price_t = \mathrm{F}(cons_t)$ et $ccons_t = \mathrm{H}(cons_t)$ est la confiance par sécurité. Elle est donc utilisée à la fois pour les équations permettant de calculer le montant de la facture et pour le calcul servant au schéma d'engagement (hachage des $cons_t$ par la fonction H).

En revanche, la confiance du fournisseur pour le calcul de $cons_t = \mathrm{S}(ECONS_t)$ s'établit par redevabilité. La confiance dans les trois autres équations ($cons'_t = \mathrm{S}(ECONS_t)$, $ccons'_t = \mathrm{H}(cons'_t)$ et $ccons'_t = \mathrm{Id}(ccons_t)$) est établie par redevabilité à partir de l'échantillon prélevé d'$ECONS_t$.

7. Il n'est pas possible d'exprimer la notion de calcul partiel dans l'architecture formelle autrement que par l'utilisation de la primitive $Spotcheck_{i,j}$.

Le résultat obtenu à l'issue de cette étape est résumé dans le Tableau 6.16.

Calcul	Confiance de U	Confiance de P
$fee = \sum_t (price_t)$	Calcul	Sécurité
$price_t = \mathrm{F}(cons_t)$	Calcul	Sécurité
$cons_t = \mathrm{S}(ECONS_t)$	Attestation	Redevabilité
$cons'_t = \mathrm{S}(ECONS_t)$		Redevabilité
$ccons_t = \mathrm{H}(cons_t)$		Sécurité
$ccons'_t = \mathrm{H}(cons'_t)$		Redevabilité
$ccons'_t = \mathrm{Id}(ccons_t)$		Redevabilité

Tableau 6.16 – Types de confiance pour Ω_3.

L'interface de l'outil *CAPRIV* à l'issue de cette étape est montrée en Figure 6.18. Elle comprend les choix de localisation et de types de confiance.

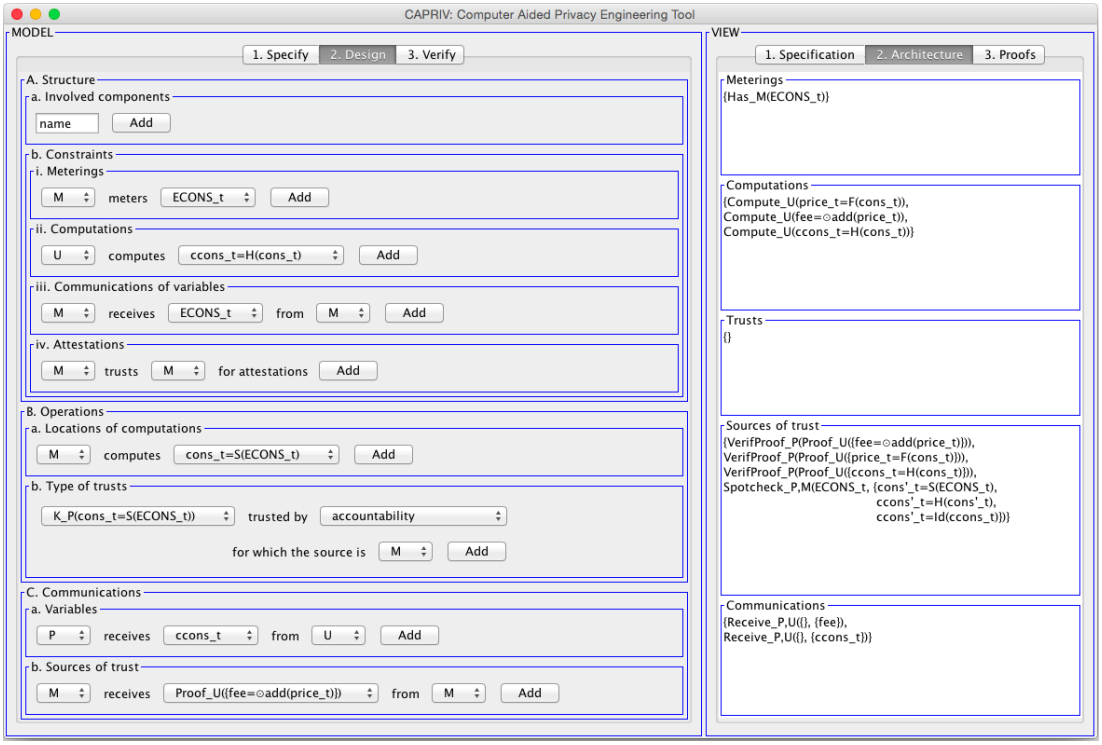

Figure 6.18 – Phase de conception dans *CAPRIV* pour Ω_3.

La Figure 6.19 représente le modèle formel obtenu à l'issue de la phase de conception de cette troisième solution. Il se distingue notamment du modèle des première (voir Figure 6.5) et deuxième (voir Figure 6.12) solutions par le retrait du lien de confiance par attestation du fournisseur vers l'usager et l'ajout de l'utilisation de la confiance par redevabilité.

L'architecture obtenue pour cette troisième solution est ensuite vérifiée comme détaillé dans la section suivante.

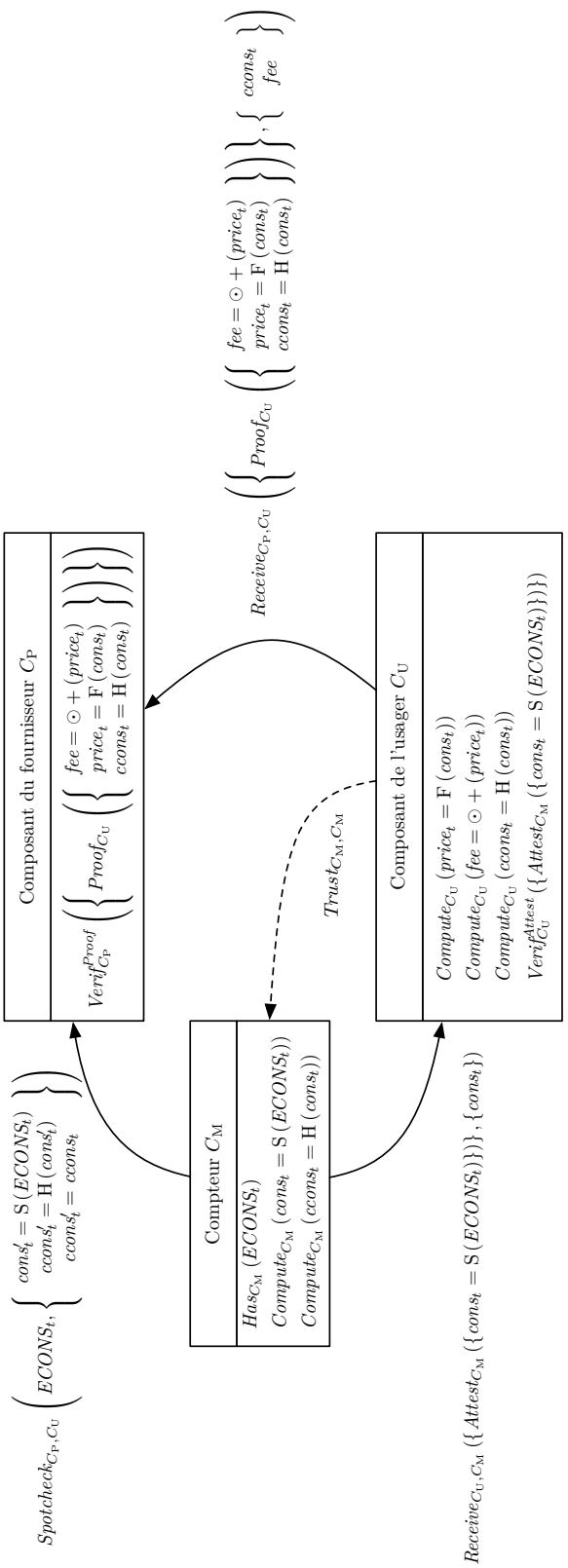

FIGURE 6.19 – Représentation dans le langage $\mathcal{L}_{FPA}$ de l'architecture pour Ω_3.

6.4.3 Vérification

Le résultat de la vérification des exigences de confidentialité pour l'usager reste toujours le même que pour la première solution. Les règles utilisées pour prouver la satisfaction de ces exigences change cependant pour le fournisseur comme illustré dans le Tableau 6.17. La règle (H4) correspond à l'utilisation de la primitive $Spotcheck_{C_P, C_M}$ sur $ECONS_t$ dans l'architecture et la règle (H6) à l'utilisation de la relation de dépendance *Dep* sur les variables tableau dont un élément est connu. Nous pouvons noter que la propriété de non-inversibilité de la fonction H est nécessaire afin que le fournisseur, qui a accès aux $ccons_t$, ne puisse pas retrouver les $cons_t$ (à travers la relation *Dep*).

Variables	Accès autorisé	Accès interdit (sauf un)
$ECONS_t$	(C_U, [H2, H5])	(C_P, [H4])
$cons_t$	(C_U, [H2])	(C_P, [H4, H6])
$price_t$	(C_U, [H3])	(C_P, [H4, H6])
fee	(C_U, [H3]), (C_P, [H2])	
$cons'_t$	(C_U, [H2, H5])	(C_P, [H4, H6])
$ccons_t$	(C_U, [H3]), (C_P, [H2])	
$ccons'_t$	(C_U, [H3, H5]), (C_P, [H2, H5])	

Tableau 6.17 – Vérification des exigences de confidentialité pour Ω_3.

La Figure 6.20 montre une capture d'écran de l'outil *CAPRIV* pour la vérification de la propriété $Has_P^{one}(price_t)$ (accès interdit sauf un). *CAPRIV* indique que la satisfaction de cette exigence est prouvée par application successive des règles H4 et H6 à l'architecture.

Le Tableau 6.18 montre de la même manière la satisfaction des propriétés d'intégrité. Les règles utilisées sont les mêmes pour l'assurance de C_U. En revanche, les règles utilisées pour justifier de la confiance de C_P sont différentes. En effet, C_P a confiance dans la mesure S par l'application des règles (K3), (KB), (B) et (B▸). Les deux autres relations sont justifiées par application des règles (K3) et (KB). L'assurance faible sur les quatre équations introduites pour modéliser la confiance par redevabilité sont vérifiées par l'application de la règle *B* à l'exception de l'équation $ccons'_t = \mathrm{H}(cons'_t)$ qui repose sur (K3) et (KB).

Relations	Connaissance	Croyance
$fee = \sum_t (price_t)$	(C_U, [K1])	(C_P, [K3, KB])
$price_t = \mathrm{F}(cons_t)$	(C_U, [K1])	(C_P, [K3, KB])
$cons_t = \mathrm{S}(ECONS_t)$	(C_U, [K5])	(C_P, [K3, KB, B, B▸])
$cons'_t = \mathrm{S}(ECONS_t)$		(C_P, [B])
$ccons_t = \mathrm{H}(cons_t)$		(C_P, [B])
$ccons'_t = \mathrm{H}(cons'_t)$		(C_P, [K3, KB])
$ccons'_t = \mathrm{Id}(ccons_t)$		(C_P, [B])

Tableau 6.18 – Vérification des exigences d'intégrité pour Ω_3.

Une capture d'écran de Why3 avec une théorie générée par *CAPRIV* pour la vérification de la propriété $B_P(cons = \mathrm{S}(ECONS_t))$ est donnée par la Figure 6.21. La théorie automatiquement

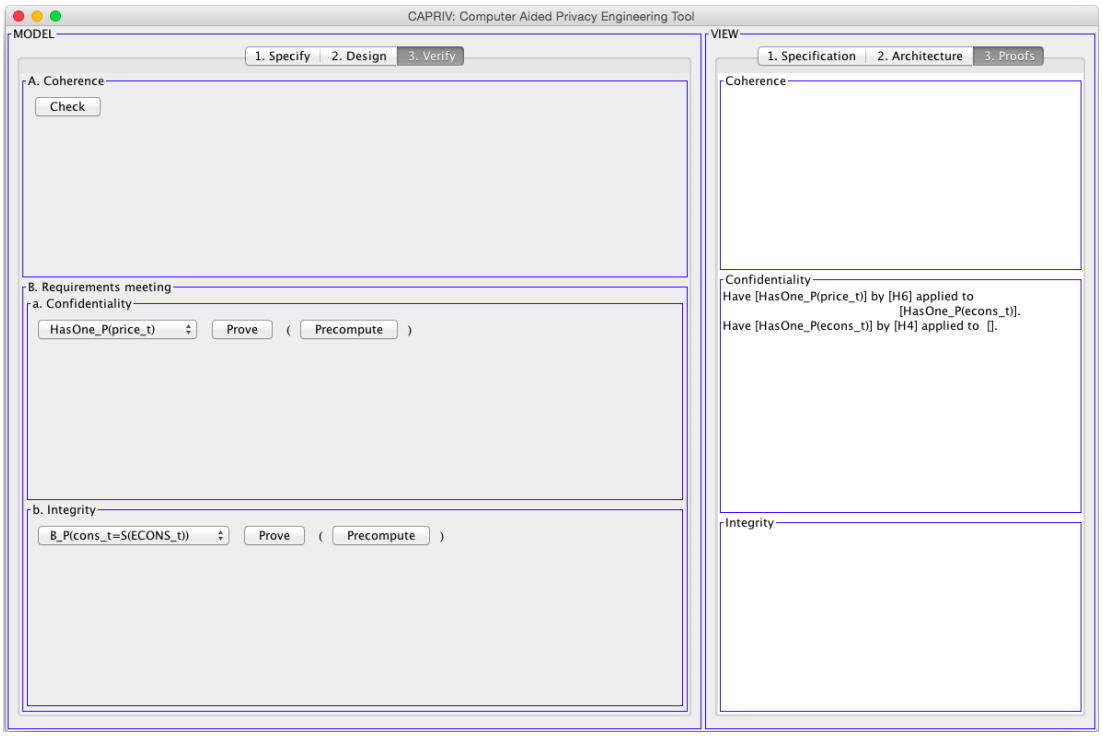

FIGURE 6.20 – Vérification de $Has_{\mathrm{P}}^{one}\ (price_t)$ dans *CAPRIV* pour Ω_3.

générée par *CAPRIV* et transmise à Why3 comprend un axiome d'injectivité (`h_inj`) pour la fonction H. Les autres équations apparaissent comme axiomes car elles sont utilisées pour permettre à P d'inférer $cons = \mathrm{S}\ (ECONS_t)$ dans le cadre de l'application de la règle (B▷).

Cette troisième solution satisfait les exigences de confidentialité et d'intégrité relaxées telles que définies dans la spécification. Elle repose à la fois sur la confiance par redevabilité pour ce qui est de la mesure S et sur la confiance par sécurité pour la tarification F et l'agrégation $\sum_t$. Le fournisseur peut vérifier le fonctionnement des compteurs en confrontant un relevé d'échantillon à des valeurs préalablement engagées.

Les trois solutions présentées dans cette section se concentrent sur une application de facturation dans le domaine du relevé intelligent de consommation électrique. Cette application et ce domaine ne sont cependant pas les seuls à bénéficier de la démarche. Le choix de la localisation des calculs et des types de confiance adéquats permet en effet de caractériser un grand nombre d'architectures de domaines et applications différents comme nous le décrivons dans la section suivante.

6.5 Conclusion

Le relevé intelligent de consommation électrique est une technologie en pleine expansion. Elle fait intervenir de multiples acteurs et est basée sur des données personnelles sensibles.

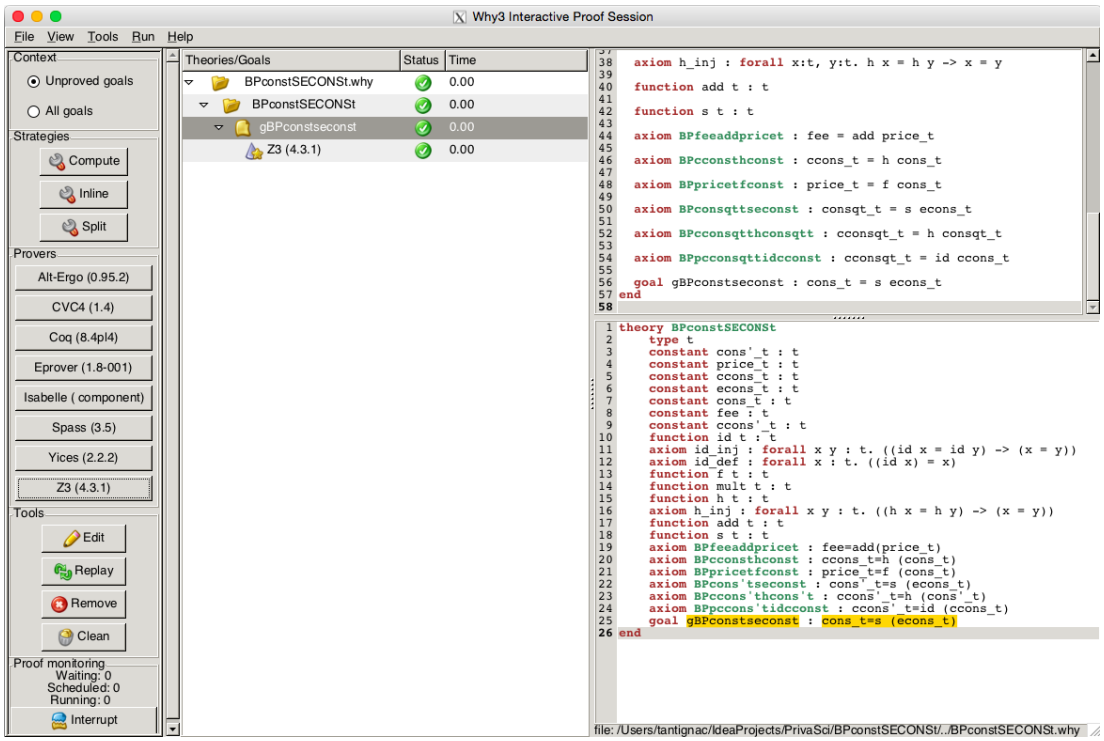

Figure 6.21 – Vérification de $K_{\mathrm{P}}\,(cons_t = \mathrm{S}\,(ECONS_t))$ dans *Why3* pour Ω_3.

Les architectures utilisables pour développer ce service sont multiples. Nous avons appliqué la démarche systématique proposée dans le Chapitre 3 pour en concevoir plusieurs. L'approche itérative a conduit à détailler différentes parties de l'architecture et à relaxer la spécification en fonction des types de confiance choisis. La démarche a été suivie en utilisant l'outil présenté dans le Chapitre 5. Son interface d'entrée et de visualisation a été conçue pour correspondre aux étapes de la démarche. La phase de vérification s'est appuyée sur le cadre formel défini dans le Chapitre 4. Nous avons aussi illustré l'intégration transparente du formalisme dans l'outil pour permettre à un concepteur non-expert en méthodes formelles de vérifier la satisfaction des exigences.

CHAPITRE 7

Conclusion

Nous présentons pour conclure une synthèse de nos contributions suivie de perspectives de recherche à court et à long terme. Enfin, nous ouvrirons notre réflexion vers la dimension économique.

7.1 Synthèse des contributions

L'état de l'art développé dans le Chapitre 2 a montré le besoin d'un cadre de développement pour le *PbD*. En effet, de nombreux travaux existants ont porté sur les *PETs* dans des applications diverses. En revanche, il n'existe que peu de méthodes pour aider un concepteur à concevoir un système et à vérifier ses propriétés. Le niveau architectural est le plus adapté en raison de ses qualités structurantes. Son abstraction permet au concepteur de raisonner à haut niveau afin d'effectuer des choix fondamentaux qui ont un impact important sur le système. Cette modélisation ouvre aussi la voie à la vérification formelle de ces propriétés.

La première contribution [ALM15] est la démarche systématique proposée dans le Chapitre 3. Elle guide le concepteur dans les phases de spécification, de conception et de vérification d'un système. Ces phases ont été ordonnées de manière à restreindre progressivement l'espace de conception. La spécification des exigences consiste à modéliser le service sous une forme équationnelle puis à définir les propriétés de confidentialité et d'intégrité attendues. La conception de l'architecture aboutit à une architecture contrainte par les choix imposés par le contexte. Les étapes les plus importantes sont la localisation des calculs, le choix des types de confiance associés et l'ajout des canaux de communication. Enfin, la phase de vérification permet au concepteur de vérifier la cohérence de l'architecture et la satisfaction des exigences et des contraintes.

La deuxième contribution [ALM14a] est le modèle formel défini dans le Chapitre 4. Il permet au concepteur d'obtenir des garanties fortes sur les propriétés du système à partir de son architecture. L'utilisation de ce modèle en suivant la démarche systématique constitue un cadre qui renforce la confiance du concepteur dans ses choix. De façon cohérente avec la démarche, les exigences de confidentialité et d'intégrité sont traitées de manière orthogonale.

Une axiomatique a été proposée afin de faciliter le raisonnement sur les architectures. Sa correction, sa complétude et sa décidabilité ont été prouvées par rapport à une sémantique à base de traces. Cette sémantique représente l'évolution de l'espace de connaissances des composants au fil de leur exécution.

La troisième contribution est l'outil qui implémente la démarche systématique et le cadre formel comme présenté dans le Chapitre 5. Cet outil guide le concepteur non-expert dans l'application de la démarche et l'utilisation d'outils de vérification. L'interface graphique de l'outil permet au concepteur de ne pas être exposé au formalisme mathématique. Une partie de la vérification est effectuée par appel à une plateforme externe de prouveurs de théorèmes. Ceci montre que la conception de systèmes en suivant une démarche de *PbD* peut s'intégrer dans des environnements formels standards. L'automatisation de la phase de vérification facilite la documentation du système et constitue une base pour une évaluation en vue d'une certification (labellisation).

Enfin, le cas d'étude du Chapitre 6 illustre l'apport de ces travaux. L'apport de la démarche systématique et les garanties offertes par le cadre formel sont mis en valeur à travers l'utilisation de l'outil. Ce cas d'étude complète et étend l'exemple développé dans les trois premiers chapitres et montre comment les choix de localisation des calculs et de types de confiance permettent d'aboutir à des solutions différentes.

7.2 Perspectives

Ce travail constitue une première étape contribuant aux méthodes formelles pour le respect de la vie privée par construction. Il peut être poursuivi dans de multiples directions et la conformité d'un protocole avec une architecture a déjà été explorée [TA15]. Nous identifions dans cette section des limites à nos contributions et nous proposons des pistes à court et long termes pour y répondre.

7.2.1 Perspectives à court terme

Les principales perspectives à court terme qui ont été identifiées sont la protection des identités, l'ajout de nouveaux types de confiance et le renforcement de l'intégration aux cycles de développement standards. Nous les détaillons dans les paragraphes suivants.

Protection de l'identité. Le cadre présenté ici se concentre sur la protection des données plutôt que sur celle des identités (selon les critères énoncés en Section 2.1.1). La démarche systématique proposée prend en compte l'anonymat des canaux de communication mais le cadre formel ne permet pas de le modéliser (et donc de vérifier ses propriétés).

La prise en compte de l'identité pourrait se faire en associant l'identité du (ou des) sujet(s) concerné(s) à chaque variable. Dans le cas de fonctions agrégatives, la variable concernée serait associée à un ensemble d'identités. Des règles additionnelles permettraient de déduire les identités à associer aux nouvelles variables calculées. Par exemple, si x et y sont associées au sujet A, alors z est aussi associée à A si $z = x + y$. En revanche, si x est associée à A et y à B, alors z est associée à $\{A, B\}$ pour la même opération qui consiste en une agrégation. Enfin, il

serait possible de modéliser des fonctions anonymisantes ou pseudonymisantes. Par exemple, si x est associée à A, alors y est associée à $\perp$ (qui représente une identité inconnue) par l'opération $y = Anon(x)$. Une « pseudonymisation » consisterait à changer A par une autre identité qui lui serait liée. Ce changement serait intégré à la sémantique en ajoutant pour chaque composant un espace de connaissances $\sigma_i^{id} : Var \rightarrow \mathcal{P}(Comp \cup \{\perp\})$ dédié à l'association des identités des sujets aux variables.

Nouveaux types de confiance. Plusieurs types de confiance ont été intégrés à ce travail (voir Section 3.5.2.3). D'autres types pourraient cependant être utiles dans certaines situations.

Une extension pourrait être de rajouter un vote par majorité. Plusieurs composants pourraient être en charge d'effectuer le même calcul et d'envoyer leur résultat. La valeur retenue serait alors la moyenne ou la valeur la plus fréquente parmi les résultats envoyés. Cette piste nécessite de pouvoir établir une relation entre les variables. En effet, les contraintes de cohérence imposent qu'une variable ne soit calculée que par un seul composant. Il faut donc établir un moyen de signaler que plusieurs variables sont liées.

Une autre extension possible serait une signature de groupe. Plutôt que de limiter la relation $Trust_{i,j}$ à deux composants, elle pourrait s'établir entre un composant et un groupe de composants : $Trust_{i,\{j\}}$. Une attestation signée par un des membres de l'ensemble $\{j\}$ pourrait alors être vérifiée par i sans que i connaisse l'identité du composant qui a apposé sa signature.

Cycle de développement. Les principales étapes du cycle de développement étudié concernent les phases de spécification des exigences, de conception de l'architecture et de vérification de satisfaction des exigences par l'architecture obtenue. Les autres étapes du cycle de conception mentionnées en Section 2.2.1 pourraient aussi bénéficier de l'approche du *PbD*.

Le développement des primitives architecturales, par exemple, est une étape cruciale dans le développement d'un système. Des failles à ce niveau peuvent invalider les garanties offertes par le système. Par conséquent, des outils adaptés devraient être développés. Par exemple, des outils d'annotation de code pourraient être utilisés pour vérifier à la compilation si un programme satisfait certaines exigences. De tels outils existent pour la sécurité [BDS13] et pourraient être étendus au respect de la vie privée. Dans l'idéal, ces annotations devraient être automatiquement générées à partir des exigences et de l'architecture. Un cycle de développement plus intégré serait donc obtenu en concordance avec l'approche du *PbD*.

Une autre difficulté couramment rencontrée lors du développement de systèmes est la difficulté due à des exigences changeant rapidement. Des méthodes de développement agiles ont été conçues pour, entre autres, répondre à ce type de contrainte. Celles-ci pourraient être analysées à l'aune du principe de minimisation. En effet, l'évolution rapide des exigences a des impacts sur la collecte et le traitement des données personnelles. Le consentement du sujet peut devoir être à nouveau demandé. Cela a des impacts en terme d'utilisabilité et risque de dégrader l'expérience de l'utilisateur.

7.2.2 Perspectives à long terme

Les résultats présentés dans ce document ne couvrent évidemment pas tous les besoins identifiés en matière de respect de la vie privée en informatique.

De nombreuses directions de recherche sont ainsi possibles. Nous en présentons trois qui consistent en une extension des notions juridiques prises en compte, une approche quantitative dans le cadre logique et l'élargissement à des applications plus complexes.

Exigences légales. Le travail présenté dans cette thèse repose sur une partie seulement des critères légaux de respect de la vie privée : la minimisation des données collectées et la correction de celles-ci. L'article 5 du projet de *GDPR* évoqué dans la Section 1.1.3 comprend d'autres critères comme l'équité et la transparence des traitements.

Des travaux sur la conformité du traitement pourraient par exemple reposer sur l'analyse de code de programmes. Des travaux existent dans le domaine de la sécurité en utilisant le principe de non-interférence [BS10] mais ne ciblent pas spécifiquement le respect de la vie privée. Dans notre cas, il serait possible d'analyser la conformité des traitements envisagés dès la spécification. L'équité est une notion plus complexe car subjective. À l'étape de la spécification, prendre en compte l'équité entre les utilisateurs pourrait permettre d'arbitrer quand plusieurs exigences sont en conflit. Nos travaux pourraient donc être étendus pour bénéficier de ces apports. Enfin, le critère de la transparence des algorithmes est un sujet d'actualité [Con14]. La transparence constitue un sujet de recherche pour les juristes [RB13] et les philosophes [Hil12]. Il est difficile de savoir quels sont les traitements appliqués aux données personnelles. Cette information peut même être légalement protégée sous couvert du secret d'affaires. Savoir comment les données du sujet sont traitées une fois collectées lui apporterait des informations. L'intérêt pour ce dernier serait de pouvoir savoir (ou même prévoir) ce qui peut être inféré sur lui. Pour ce faire, il devrait considérer les hypothèses les plus pessimistes (pour le sujet) sur les capacités de déduction des composants ayant collecté ses données et faire des hypothèses (réalistes ou pessimistes) sur leurs connaissances auxiliaires.

L'intégration de ces travaux nécessiterait des modifications importantes du cadre formel pour pouvoir raisonner sur des politiques de gestion de données. De plus, ces travaux se rapprocheraient du *PpP* [1] en ce qu'ils se concentreraient davantage sur l'information et le choix de l'utilisateur selon la terminologie d'usage [SC09].

Approche quantitative. L'approche adoptée dans ce travail est principalement qualitative. Certaines spécifications pourraient requérir davantage de souplesse et nécessiter une approche quantitative.

Un moyen serait de définir des primitives permettant d'utiliser des critères de respect de la vie privée tels que ceux mentionnés dans la Section 2.1.1. Le critère ϵ-différentiel semble particulièrement indiqué pour sa robustesse face aux opérations de composition.

Par ailleurs, des calculs successifs peuvent être effectués auprès de plusieurs composants pour satisfaire un service. La confiance accordée par les composants intéressés dépend alors de la

1. *Privacy by Policy*, respect de la vie privée par politique.

chaîne de confiance. Des modèles quantitatifs ont été proposés [LD08 ; Alh+14] et pourraient servir de base pour définir un niveau de confiance pour l'intégrité des propriétés établies.

Applications. Les applications présentées ici sont des applications de service exprimables sous forme équationnelle. Les opérations d'agrégation sont donc utilisables et permettent de limiter les divulgations inutiles. D'autres services, basés sur le triptyque « SoLoMo » [2], prennent une importance croissante. De multiples travaux s'intéressent à des applications sociales [PS14 ; FAZ09]. Ceux-ci se concentrent davantage à l'analyse des systèmes qu'à leur conception. Le même constat peut être fait pour les services géolocalisés. La récente faille de données concernant le service de voitures avec chauffeur Uber a récemment mis en lumière la criticité de ces données. De nombreux travaux concernent ce domaine [GKPC10] et proposent des solutions adaptées [Dam13]. La mobilité de ces services entraîne leur ubiquité et la continuité des traces numériques laissées par les utilisateurs. Les services mobiles sont aussi source d'importants risques pour le respect de la vie privée [Arf+14].

À l'ère de la « personnalisation de masse » [3], l'interconnexion de toutes ces applications va continuer de croître fortement. Ce mouvement a déjà commencé avec les réseaux sociaux proposant des authentifications par le système *OAuth*. Il devrait continuer avec les plateformes d'interfaces comme *IFTTT* [4] ou Zapier. Ces applications posent des problèmes importants en terme de respect de la vie privée tant les données sont concentrées et les usages complexes. Des approches spécifiques à chaque application rendraient plus efficace la démarche du *PbD*.

7.3 Dimension économique

Le respect de la vie privée est un sujet intrinsèquement pluridisciplinaire. Nous nous sommes concentrés jusqu'ici sur ses aspects informatique et juridique. Il existe cependant d'autres facettes à explorer parmi lesquelles l'interaction avec l'économie.

Comme nous l'avons vu en introduction, l'économie est une source de pression importante sur le respect de la vie privée. Deux catégories de modèles d'affaires se distinguent comme l'a rappelé dernièrement une lettre ouverte de Tim Cook, directeur général exécutif d'Apple, adressée aux clients de la marque : « Notre modèle d'affaires est vraiment simple : nous vendons des produits géniaux. Nous ne construisons pas de profils basés sur le contenu de vos courriers électroniques ou sur vos habitudes de navigation pour les vendre à des annonceurs. » [5]. Cette position (qu'elle corresponde à la réalité ou pas) est plus facile à promouvoir pour une entreprise dont la majeure partie des marges est réalisée par la vente de produits physiques. Il est difficile pour des entreprises comme Facebook ou Google d'adopter une stratégie aussi respectueuse

2. Social, local et mobile. De nombreuses applications et services *Web* se concentrent sur des solutions mêlant ces trois aspects.

3. *Mass customization* en anglais.

4. *If This Then That* est un service permettant de déclencher des actions (comme envoyer vers le compte Dropbox ou envoyer un e-mail) lors de l'occurrence d'événements (comme une mention sur Twitter ou l'activation d'un objet connecté).

5. « *Our business model is very straightforward : We sell great products. We don't build a profile based on your email content or web browsing habits to sell to advertisers.* »

des données personnelles car leurs utilisateurs [6] semblent réticents à payer pour des services virtuels.

Par conséquent, nous pensons que l'interaction entre économie et respect de la vie privée en informatique est un champ de recherche prometteur. Des recherches ont déjà été menées dans cette direction. La valorisation des données issues de l'auto-quantification [7] [Li+13], les modèles d'affaires respectueux de la vie privée [Liu+11], ou encore le développement des crypto-monnaies [HM14] vont dans ce sens.

Les modèles d'affaires de demain devront aligner l'intérêt des utilisateurs et celui des fournisseurs. L'informatique a son rôle à jouer par l'apport de nouvelles techniques qui rendent possibles de tels modèles. Pour y parvenir, le développement de démarches et d'outils supportant le *PbD* est une étape nécessaire pour rendre plus systématique le respect de la vie privée des utilisateurs.

6. Les clients de Facebook et Google sont ceux qui paient leurs services, ce ne sont donc pas les internautes *lambda* mais en majorité les annonceurs.

7. *Quantified self* en anglais.

Bibliographie

[Abr96] Jean-Raymond Abrial. *The B-Book*. Cambridge University Press, 1996.

[AF01] Martín Abadi et Cédric Fournet. “Mobile Values, New Names, and Secure Communication”. In : *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’01. London, United Kingdom : ACM, 2001, p. 104–115.

[AGK03] John Argyrakis, Stefanos Gritzalis et Chris Kioulafas. “Privacy enhancing technologies : A review”. In : *Electronic Government*. Springer, 2003, p. 282–287.

[Alh+14] Nagham Alhadad, Yann Busnel, Patricia Serrano-Alvarado et Philippe Lamarre. “Trust Evaluation of a System for an Activity with Subjective Logic”. English. In : *Trust, Privacy, and Security in Digital Business*. Sous la dir. de Claudia Eckert, SokratisK. Katsikas et Günther Pernul. T. 8647. Lecture Notes in Computer Science. Springer International Publishing, 2014, p. 48–59.

[All97] Robert J. Allen. “A Formal Approach to Software Architecture”. Thèse de doct. University Pittsburgh, PA 15213 : School of Computer Science Carnegie Mellon, 1997.

[ALM12] Thibaud Antignac et Daniel Le Métayer. “PrivaSy : systematic exploration of the design space using constraint solving techniques (ongoing work)”. Atelier de Protection de la Vie Privée (APVP 2012), Île de Groix, France. 2012.

[ALM14a] Thibaud Antignac et Daniel Le Métayer. “Privacy Architectures : Reasoning about Data Minimisation and Integrity”. English. In : *Security and Trust Management*. Sous la dir. de Sjouke Mauw et Christian Damsgaard Jensen. T. 8743. Lecture Notes in Computer Science. Springer, 2014, p. 17–32.

[ALM14b] Thibaud Antignac et Daniel Le Métayer. “Privacy by Design : From Technologies to Architectures”. English. In : *Privacy Technologies and Policy*. Sous la dir. de Bart Preneel et Demosthenes Ikonomou. T. 8450. Lecture Notes in Computer Science. Springer, 2014, p. 1–17.

[ALM15] Thibaud Antignac et Daniel Le Métayer. “Trust Driven Strategies for Privacy by Design”. English. In : *Trust Management IX*. Sous la dir. de Christian Damsgaard Jensen, Stephen Marsh, Theo Dimitrakos et Yuko Murayama. T. 454. IFIP Advances in Information and Communication Technology. Springer International Publishing, 2015, p. 60–75.

[And08] Ross J. Anderson. *Security Engineering : A Guide to Building Dependable Distributed Systems*. 2e éd. Wiley, 2008.

[And11] Marc Andreessen. "Why Software Is Eating the World". In : *The Wall Street Journal* (2011).

[Arf+14] Ghada Arfaoui, Sébastien Gambs, Patrick Lacharme, Jean-Francois Lalande, Roch Lescuyer et Jean-Claude Paillès. "A Privacy-Preserving Contactless Transport Service for NFC Smartphones". English. In : *Mobile Computing, Applications, and Services*. Sous la dir. de Gérard Memmi et Ulf Blanke. T. 130. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer International Publishing, 2014, p. 282–285.

[Bal+10] Josep Balasch, Alfredo Rial, Carmela Troncoso et Christophe Geuens. "PrETP : Privacy-Preserving Electronic Toll Pricing". In : *Proceedings of the 19th USENIX Security Symposium*. Washington DC, USA, août 2010, p. 63–78.

[Bar+06] A. Barth, A. Datta, J.C. Mitchell et H. Nissenbaum. "Privacy and contextual integrity : framework and applications". In : *Security and Privacy, 2006 IEEE Symposium on*. 2006, 15 pp. –198.

[BCC88] Gilles Brassard, David Chaum et Claude Crepeau. "Minimum Disclosure Proofs of Knowledge". In : *Journal of Computer and System Sciences* 37 (1988), p. 156–189.

[BCK12] Len Bass, Paul Clements et Rick Kazman. *Software Architecture in Practice*. 3rd. SEI series in Software Engineering. Addison-Wesley, 2012.

[BDK04] M. Backes, M. Diirmuth et G. Karjot. "Unification in privacy policy evaluation - translating EPAL into Prolog". In : *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*. 2004, p. 185–188.

[BDS13] Niklas Broberg, Bart van Delft et David Sands. "Paragon for Practical Programming with Information-Flow Control". English. In : *Programming Languages and Systems*. Sous la dir. de Chung-chieh Shan. T. 8301. Lecture Notes in Computer Science. Springer International Publishing, 2013, p. 217–232.

[Bec+01] K. Beck, M. Beedle, A. van Bennekum et et al. *Manifesto for Agile Software Development*. 2001.

[Bei+14] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard et Anat Paskin-Cherniavsky. "Non-Interactive Secure Multiparty Computation". English. In : *Advances in Cryptology – CRYPTO 2014*. Sous la dir. de Juan A. Garay et Rosario Gennaro. T. 8617. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2014, p. 387–404.

[BFM88] Manuel Blum, Paul Feldman et Silvio Micali. "Non-interactive Zero-knowledge and Its Applications". In : *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. New York, NY, USA : ACM, 1988, p. 103–112.

[BHF04] Bob Blakley, Craig Heath et The Open Group Security Forum. *Security Design Patterns*. Rapp. tech. The Open Group, 2004.

[BKP12] Alastair R Beresford, Dorothea Kübler et Sören Preibusch. "Unwillingness to pay for privacy : A field experiment". In : *Economics Letters* 117.1 (2012), p. 25–27.

[BMB11] Moritz Y. Becker, Alexander Malkis et Laurent Bussard. "A Practical Generic Privacy Language". In : *Information Systems Security*. T. 6503. Lecture Notes in Computer Science. Springer, 2011, p. 125–139.

[Bob+08] François Bobot, Sylvain Conchon, E Contejean, Mohamed Iguernelala, Stéphane Lescuyer et Alain Mebsout. "The Alt-Ergo automated theorem prover". In : (2008).

[Bob+11] François Bobot, Jean-Christophe Filliâtre, Claude Marché et Andrei Paskevich. "Why3 : Shepherd your herd of provers". In : *Boogie 2011 : First International Workshop on Intermediate Verification Languages*. 2011, p. 53–64.

[Boe88] B.W. Boehm. "A spiral model of software development and enhancement". In : *IEEE Computer* 21.5 (1988), p. 61–72.

[BRJ05] Grady Booch, James Rumbaugh et Ivar Jacobson. *Unified Modeling Language User Guide*. Sous la dir. d'Addison-Wesley. 2e éd. Object Technology Series. Pearson, 2005.

[BS10] Niklas Broberg et David Sands. "Paralocks : Role-based Information Flow Control and Beyond". In : *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '10. New York, NY, USA : ACM, 2010, p. 431–444.

[Byg02] Lee A Bygrave. "Privacy-enhancing technologies-caught between a rock and a hard place". In : *Privacy Law and Policy Reporter* 9 (2002), p. 135–137.

[Can01] Ron Canetti. "Universally Composable Security : A New Paradigm for Cryptographic Protocols". In : *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. Sous la dir. de Ron Canetti. 2001.

[Cav09] Ann Cavoukian. *Privacy by Design . . . Take the Challenge*. 2009.

[CC13] Pierre Collin et Nicolas Colin. *Mission d'expertise sur la fiscalité de l'économie numérique*. 2013.

[CD09] Véronique Cortier et Stéphanie Delaune. "Safely Composing Security Protocols". In : *Formal Methods in System Design* 34.1 (2009), p. 1–36.

[CF14] CNIL et FIEEC. "Pack de conformité pour les compteurs communicants". 2014.

[Cha81] David L. Chaum. "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms". In : *Commun. ACM* 24.2 (fév. 1981), p. 84–90.

[Cha85] David Chaum. "Security without identification : transaction systems to make big brother obsolete". In : *Communications of the ACM* 28 (1985), p. 1030–1044.

[Cho+98] Benny Chor, Eyal Kushilevitz, Oded Goldreich et Madhu Sudan. "Private information retrieval". In : *Journal of the ACM* 45.6 (1998), p. 965–981.

[Cle96] Paul C. CLEMENTS. “A Survey of Architecture Description Languages”. In : *Proceedings of the 8th International Workshop on Software Specification and Design*. Washington, DC, USA : IEEE Computer Society, 1996, p. 16.

[Con14] CONSEIL D'ETAT. “Etude annuelle 2014 du Conseil d'Etat - Le numérique et les droits fondamentaux”. La Documentation française. 2014.

[Con50] CONSEIL DE L'EUROPE. “Convention de sauvegarde des Droits de l'Homme et des Libertés fondamentales telle qu'amendée par les Protocoles n ° 11 et n ° 14”. 1950.

[Dam13] Maria Luisa DAMIANI. “Privacy Enhancing Techniques for the Protection of Mobility Patterns in LBS : Research Issues and Trends”. English. In : *European Data Protection : Coming of Age*. Sous la dir. de Serge GUTWIRTH, Ronald LEENES, Paul de HERT et Yves POULLET. Springer Netherlands, 2013, p. 223–239.

[Dia+09] Claudia DIAZ, Eleni KOSTA, Hannelore DEKEYSER, Markulf KOHLWEISS et Nigusse GIRMA. “Privacy preserving electronic petitions”. In : *Identity in the Information Society* 1.1 (2009), p. 203–209.

[Dir95] *Directive 2006/24/EC of the European Parliament and of the Council of 15 March 2006 on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/ECe 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. 2006.

[Dir95] *Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. 1995.

[DKR10] Stéphanie DELAUNE, Steve KREMER et Mark D. RYAN. “Verifying Privacy-Type Properties of Electronic Voting Protocols : A Taster”. In : *Towards Trustworthy Elections – New Directions in Electronic Voting*. Sous la dir. de David CHAUM, Markus JAKOBSSON, Ronald L. RIVEST, Peter Y. A. RYAN, Josh BENALOH, Mirosław KUTYŁOWSKI et Ben ADIDA. T. 6000. Lecture Notes in Computer Science. Springer, mai 2010, p. 289–309.

[DKR11] George DANEZIS, Markulf KOHLWEISS et Alfredo RIAL. *Differentially Private Billing with Rebates*. Technical report MSR-TR-2011-10. Microsoft Research, fév. 2011.

[DMB08] Leonardo DE MOURA et Nikolaj BJØRNER. “Z3 : An efficient SMT solver”. In : *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, p. 337–340.

[Dwo06] Cynthia DWORK. “Differential privacy”. In : *Automata, languages and programming*. Springer, 2006, p. 1–12.

[DY81] Danny DOLEV et Andrew C. YAO. *On the Security of Public Key Protocols*. Rapp. tech. Stanford, CA, USA, 1981.

[Erk+13] Z. Erkin, J.R. Troncoso-Pastoriza, R.L. Lagendijk et F. Perez-Gonzalez. "Privacy-preserving data aggregation in smart metering systems : an overview". In : *Signal Processing Magazine, IEEE* 30.2 (2013), p. 75–86.

[FAZ09] Philip W. L. Fong, Mohd Anwar et Zhen Zhao. "A Privacy Preservation Model for Facebook-Style Social Network Systems". In : *Computer Security – ESORICS 2009*. T. 5789. Lecture Notes in Computer Science. Springer, 2009, p. 303–320.

[FoP09] Working Party on Police and Justice. *The Future of Privacy*. Working report WP168. Joint contribution to the Consultation of the European Commission on the legal framework for the fundamental right to protection of personal data, déc. 2009.

[Fou+13] Cédric Fournet, Markulf Kohlweiss, George Danezis et Zhengqin Luo. "ZQL : A Compiler for Privacy-preserving Data Processing". In : *Proceedings of the 22Nd USENIX Conference on Security*. SEC'13. Berkeley, CA, USA : USENIX Association, 2013, p. 163–178.

[GDPR14] European Parliament. *European Parliament legislative resolution of 12 March 2014 on the proposal for a regulation of the European Parliament and of the Council on the protection of individuals with regard to the processing of personal data and on the free movement of such data*. en. General Data Protection Regulation, Ordinary legislative procedure : first reading. 2014.

[Gen10] Craig Gentry. "Computing arbitrary functions of encrypted data". In : *Communications of the ACM* 53.3 (2010), p. 97–105.

[GFC83] German Federal Constitutional Court. *Decision on national census 15.12.1983*. 1983.

[GKPC10] Sébastien Gambs, Marc-Olivier Killijian et Miguel Núñez del Prado Cortez. "Show me how you move and I will tell you who you are". In : *SPRINGL '10 : Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*. New-York, USA : ACM Press, 2010, p. 34–41.

[GMR85] Shafi Goldwasser, Silvio Micali et Charles Rackoff. "The knowledge complexity of interactive proof-systems". In : *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. STOC '85. New York, NY, USA : ACM, 1985, p. 291–304.

[Gol03] Ian Goldberg. "Privacy-enhancing technologies for the Internet, II : Five years later". In : *Privacy Enhancing Technologies*. Springer, 2003, p. 1–12.

[Gol07] Ian Goldberg. "Privacy Enhancing Technologies for the Internet III : Ten Years Later". In : *Digital Privacy : Theory, Tech., and Practices*. Sous la dir. d'Alessandro Acquisti, Stefanos Gritzalis, Costos Lambrinoudakis et Sabrina di Vimercati. Auerbach Publications, 2007, p. 3–18.

[Gor00] Mike Gordon. "From LCF to HOL : a short history." In : *Proof, Language, and Interaction*. 2000, p. 169–186.

[Gre14] Glenn Greenwald. *No Place to Hide : Edward Snowden, the NSA, and the U.S. Surveillance State*. Metropolitan Books, 2014, p. 272.

[Gru92] Werner Gruhl. “Lessons Learned, Cost/Schedule Assessment Guide”. 1992.

[GTD11] Sada Gürses, Carmela Troncoso et Claudia Diaz. *Engineering Privacy by Design*. Presented at the Computers, Privacy & Data Protection conference. Brussels, Belgium, jan. 2011.

[GWB97] Ian Goldberg, David Wagner et Eric Brewer. “Privacy-enhancing technologies for the Internet”. In : *Compcon ’97. Proceedings, IEEE*. 1997.

[Haf10] Munawar Hafiz. “A Pattern language for developing privacy enhancing technologies”. In : *Software : Practice and Experience* 43.7 (2010), p. 769–787.

[HG08] Mireille Hildebrandt et Serge Gutwirth. *Profiling the European citizen*. Springer, 2008.

[Hil12] Mireille Hildebrandt. “The Dawn of a Critical Transparency Right for the Profiling Era”. In : *Digital Enlightenment Yearbook 2012*. Sous la dir. de J. Bus, M. Crompton, M. Hildebrandt et G. Metakides. Amsterdam. IOS Press, 2012.

[HIPPA96] United States of America. “Health Insurance Portability and Accountability Act”. 1996.

[HM14] Sarah Jane Hughes et Stephen T. Middlebrook. “Regulating Cryptocurrencies in the United States : Current Issues and Future Directions”. In : *William Mitchell Law Review* 40.813 (2014).

[Hoe14] Jaap-Henk Hoepman. “Privacy Design Strategies”. English. In : *ICT Systems Security and Privacy Protection*. Sous la dir. de Nora Cuppens-Boulahia, Frédéric Cuppens, Sushil Jajodia, Anas Abou El Kalam et Thierry Sans. T. 428. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2014, p. 446–459.

[Hoh+06] B. Hoh, M. Gruteser, Hui Xiong et A. Alrabady. “Enhancing Security and Privacy in Traffic-Monitoring Systems”. In : *Pervasive Computing, IEEE* 5.4 (2006), p. 38–46.

[IL07] Muhammad Usman Iqbal et Samsung Lim. “An automated real-world privacy assessment of GPS tracking and profiling”. In : *Proceedings of the 2nd Workshop on Social Implications of National Security : From Dataveillance to Uberveillance*. 2007, p. 225–240.

[ISO09] *ISO 15408 — Common Criteria*. Rapp. tech. International Organization for Standardisation, 2009.

[ISO11] *ISO 29100 — Privacy framework*. Rapp. tech. International Organization for Standardisation, 2011.

[ITS14] Inria et TNS-Sofres. “Baromètre sur les français et le numérique”. In : (2014).

[Jaf+11] Mohammad Jafari, Philip W.L. Fong, Reihaneh Safavi-Naini, Ken Barker et Nicholas Paul Sheppard. “Towards defining semantic foundations for purpose-based privacy policies”. In : *Proceedings of the first ACM conference on Data and application security and privacy*. CODASPY ’11. New York, NY, USA : ACM, 2011, p. 213–224.

[Jes11] Tobias Jeske. “Privacy-preserving Smart Metering without a Trusted-third-party.” In : *SECRYPT* 2011 (2011), p. 114–123.

[JJ08] Wiebren de Jonge et Bart Jacobs. “Privacy-Friendly Electronic Traffic Pricing via Commits”. In : *Formal Aspects in Security and Trust*. Sous la dir. de Pierpaolo Degano, Joshua Guttman et Fabio Martinelli. T. 5491. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008, p. 143–161.

[JJK11] Marek Jawurek, Martin Johns et Florian Kerschbaum. “Plug-in privacy for smart metering billing”. In : *Privacy Enhancing Technologies'11*. Springer, 2011, p. 192–210.

[JKD12] Marek Jawurek, Florian Kerschbaum et George Danezis. *Privacy Technologies for Smart Grids - A Survey of Options*. Rapp. tech. MSR-TR-2012-119. Microsoft Research, 2012.

[KDK11] Klaus Kursawe, George Danezis et Markulf Kohlweiss. “Privacy-friendly aggregation for the smart-grid”. In : *Privacy-friendly aggregation for the smart-grid*. T. Privacy Enhancing Technologies. Springer, 2011, p. 175–191.

[Ker12] Florian Kerschbaum. *Privacy-Preserving Computation (Position Paper)*. Presented at the Annual Privacy Forum conference. Cyprus, 2012.

[Ker83] Auguste Kerckhoffs. “La cryptographie militaire”. In : *Journal des sciences militaries* (1883).

[Kun14] Antonio Kung. “PEARs : Privacy Enhancing ARchitectures”. In : *Proceedings of the Annual Privacy forum*. Greece, 2014.

[Lan+08] Peter Langendörfer, Michael Maaser, Krzysztof Piotrowski et Steffen Peter. “Privacy-Enhancing Technique : A Survey and Classification”. In : *Handbook of Research on Wireless Security*. Sous la dir. d'Yan Zhang, Jun Zheng et Miao Ma. Germany : IGI Global, 2008.

[Lan01] Marc Langheinrich. “Privacy by Design — Principles of Privacy-Aware Ubiquitous Systems”. English. In : *Ubicomp 2001 : Ubiquitous Computing*. Sous la dir. de Gregory D. Abowd, Barry Brumitt et Steven Shafer. T. 2201. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, p. 273–291.

[LD08] Emiliano Lorini et Robert Demolombe. “From Binary Trust to Graded Trust in Information Sources : A Logical Perspective”. English. In : *Trust in Agent Societies*. Sous la dir. de Rino Falcone, SuzanneK. Barber, Jordi Sabater-Mir et MunindarP. Singh. T. 5396. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, p. 205–225.

[LeM+07] Michael LeMay, George Gross, Carl A. Gunter et Sanjam Garg. “Unified Architecture for Large-Scale Attested Metering”. In : *40th annual Hawaii International Conference on System Sciences (HICSS'07)*. 2007, p. 115–124.

[Li+13] Chao Li, Daniel Yang Li, Gerome Miklau et Dan Suciu. “A Theory of Pricing Private Data”. In : *Proceedings of the 16th International Conference on Database Theory*. ICDT '13. New York, NY, USA : ACM, 2013, p. 33–44.

[Lin+12] Hsiao-Ying Lin, Wen-Guey Tzeng, Shiuan-Tzuo Shen et Bao-Shuh P Lin. "A practical smart metering system supporting privacy preserving billing and load monitoring". In : *Applied Cryptography and Network Security*. Springer, 2012, p. 544–560.

[Liu+11] Zhan Liu, Riccardo Bonazzi, Boris Fritscher et Yves Pigneur. "Privacy-friendly Business Models for Location-based Mobile". In : *J. Theor. Appl. Electron. Commer. Res.* 6.2 (août 2011), p. 90–107.

[Liu+12] Jing Liu, Yang Xiao, Shuhui Li, Wei Liang et C. L. Philip Chen. "Cyber Security and Privacy Issues in Smart Grids". In : *Communications Surveys Tutorials, IEEE* 14.4 (2012), p. 981–997.

[LLL10] Fenjun Li, Bo Luo et Peng Liu. "Secure information aggregation for smart grids using homomorphic encryption". In : *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*. IEEE, 2010, p. 327–332.

[LLV07] Ninghui Li, Tiancheng Li et Suresh Venkatasubramanian. "t-Closeness : Privacy Beyond k-Anonymity and l-Diversity." In : *ICDE'07*. 2007, p. 106–115.

[LM09] Daniel Le Métayer. "A Formal Privacy Management Framework". In : *Formal Aspects in Security and Trust*. Sous la dir. de Pierpaolo Degano, Joshua Guttman et Fabio Martinelli. T. 5491. Lecture Notes in Computer Science. Inria Grenoble Rhône-Alpes 655 venue de l'Europe Montbonnot France. Malaga : Springer Berlin / Heidelberg, 2009, p. 162–176.

[LM10] Daniel Le Métayer. "Privacy by Design : A Matter of Choice". In : *Data Protection in a Profiled World*. Sous la dir. de Serge Gutwirth, Yves Poullet et Paul De Hert. 10.1007/978-90-481-8865-9_20. Springer Netherlands, 2010, p. 323–334.

[LM13] Daniel Le Métayer. "Privacy by Design : A Formal Framework for the Analysis of Architectural Choices". In : *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*. CODASPY '13. New York, NY, USA : ACM, 2013, p. 95–104.

[Loi78] *Loi n° 78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés*. 1978.

[LSP07] Thierry Lecomte, Thierry Servat et Guilhem Pouzancre. "Formal methods in safety-critical railway systems." In : *Proceedings of SMBF 2007*. Ouro Preto, 2007.

[Mac+07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke et Muthuramakrishnan Venkitasubramaniam. "l-diversity : Privacy beyond k-anonymity". In : *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), p. 3.

[Man+11] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh et Angela H Byers. *Big data : The next frontier for innovation, competition, and productivity*. 2011.

[Man+13] Vassilis Manousakis, Christos Kalloniatis, Evangelia Kavakli et Stefanos Gritzalis. “Privacy in the Cloud : Bridging the Gap between Design and Implementation”. In : *Advanced Information Systems Eng. Workshops*. Sous la dir. de Xavier Franch et Pnina Soffer. T. 148. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2013, p. 455–465.

[MM+10] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet et David Irwin. “Private memoirs of a smart meter”. In : *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building, BuildSys ’10*. ACM. 2010, p. 61–66.

[Mon76] J. Donald Monk. *Mathematical Logic*. Springer-Verlag, 1976.

[MS08] *Privacy Guidelines for Developing Software Products and Services*. Rapp. tech. Microsoft, 2008.

[Not+15] Nicolás Notario, Alberto Crespo, Daniel Le Métayer, José M. Del Álamo, Yod Samuel Martín García, Inga Kroener, Thibaud Antignac et David Wright. “PRIPARE : Integrating Privacy Best Practices into a Privacy Engineering Methodology”. In : *IWPE 2015*. (to appear). IEEE, 2015.

[OASIS14] *Privacy by Design Documentation for Software Engineers*. Rapp. tech. Organization for the Advancement of Structured Information Standards, 2014.

[Odl03] Andrew Odlyzko. “Privacy, economics, and price discrimination on the Internet”. In : *Proceedings of the 5th international conference on Electronic commerce*. ACM, 2003, p. 355–366.

[OEC80] OECD. *Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*. Rapp. tech. 1980.

[OSV10] Martin Odersky, Lex Spoon et Bill Venners. *Programming in Scala*. Walnut Creek, Calif. : Artima, 2010, p. LI, 852.

[Pai99] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. English. In : *Advances in Cryptology — EUROCRYPT ’99*. Sous la dir. de Jacques Stern. T. 1592. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1999, p. 223–238.

[PB11] Siani Pearson et Azzedine Benameur. “A Decision Support System for Design for Privacy”. In : *Privacy and Identity Management for Life*. Sous la dir. de Simone Fischer-Hübner, Penny Duquenoy, Marit Hansen, Ronald Leenes et Ge Zhang. T. 352. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2011, p. 283–296.

[PBB09] Raluca Ada Popa, Hari Balakrishnan et Andrew J. Blumberg. “VPriv : Protecting Privacy in Location-Based Vehicular Services”. In : *Proceedings of the 18th USENIX Security Symposium*. Montreal, Canada, août 2009, p. 335–350.

[Pet10] Ronald Petrlic. “A privacy-preserving concept for smart grids”. In : *Sicherheit in vernetzten Systemen* 18 (2010), B1–B14.

[PH10] Andreas Pfitzmann et Marit Hansen. “A terminology for talking about privacy by data minimization : Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management”. V0.34. 2010.

[Pos73] Richard A Posner. *Economic analysis of law*. Little Brown et Company, 1973.

[PS14] Raúl Pardo et Gerardo Schneider. “A Formal Privacy Policy Framework for Social Networks”. English. In : *Software Engineering and Formal Methods*. Sous la dir. de Dimitra Giannakopoulou et Gwen SalaÃ¼n. T. 8702. Lecture Notes in Computer Science. Springer International Publishing, 2014, p. 378–392.

[Puc04] Riccardo Pucella. “Deductive Algorithmic Knowledge”. In : *CoRR* cs.AI/0405038 (2004).

[Rab81] Michael O. Rabin. *How to exchange secrets by oblivious transfer, A, 1981*. Rapp. tech. 1981.

[RAD78] Ronald L. Rivest, Len Adleman et Michael L. Dertouzos. “On data banks and privacy homomorphisms”. In : *Foundations of Secure Computation* (1978), p. 169–177.

[RB13] Antoinette Rouvroy et Thomas Berns. “Gouvernementalité algorithmique et perspectives d’émancipation : le disparate comme condition d’individuation par la relation ?” In : *RESEAUX* 31.177 (2013), p. 163–196.

[RCRC73] United States Department of Health, Education and Welfare. *Records, Computers and the Rights of Citizens*. Rapp. tech. 1973.

[RD10] Alfredo Rial et George Danezis. *Privacy-Preserving Smart Metering*. Technical report MSR-TR-2010-150. Microsoft Research, nov. 2010.

[RP09] Antoinette Rouvroy et Yves Poullet. “The right to informational self-determination and the value of self-development : Reassessing the importance of privacy for democracy”. In : *Reinventing Data Protection ?* Springer, 2009, p. 45–76.

[RR98] Michael K. Reiter et Aviel D. Rubin. “Crowds : Anonymity for Web Transactions”. In : *ACM Trans. Inf. Syst. Secur.* 1.1 (nov. 1998), p. 66–92.

[RSG98] M.G. Reed, P.F. Syverson et D.M. Goldschlag. “Anonymous connections and onion routing”. In : *Selected Areas in Communications, IEEE Journal on* 16.4 (1998), p. 482–494.

[SB10] Christoph Sprenger et David Basin. “Developing Security Protocols by Refinement”. In : *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS ’10. New York, NY, USA : ACM, 2010, p. 361–374.

[SC09] Sarah Spiekermann et Lorrie Faith Cranor. “Engineering Privacy”. In : *IEEE Transactions on Software Engineering* 35.1 (2009), p. 67–82.

[Sch+05] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann et Peter Sommerlad. *Security Patterns : Integrating Security and Systems Engineering*. Wiley, 2005.

[Sch07] Bruce Schneier. *Applied cryptography : protocols, algorithms, and source code in C*. john wiley & sons, 2007.

[Sch42] Joseph Alois Schumpeter. *Socialism, capitalism and democracy*. Harper et Brothers, 1942.

[Sch96] Bruce Schneier. *Applied Cryptography*. 2e éd. Wiley, 1996.

[Sha79] Adi Shamir. "How to share a secret". In : *Communications of the ACM* 22.11 (1979), p. 612–613.

[Sol06] Daniel J Solove. "A taxonomy of privacy". In : *University of Pennsylvania Law Review* (2006), p. 477–564.

[Som11] Ian Sommerville. *Software engineering*. 9th edition, international edition. Boston : Pearson, 2011.

[Son99] Dawn Xiaodong Song. "Athena : A New Efficient Automatic Checker for Security Protocol Analysis". In : *Proceedings of the 12th IEEE Workshop on Computer Security Foundations*. CSFW '99. Washington, DC, USA : IEEE Computer Society, 1999, p. 192.

[SPP01] Dawn Song, Adrian Perrig et Doantam Phan. "AGVI — Automatic Generation, Verification, and Implementation of Security Protocols". In : *Proceedings of 13th Conference on Computer Aided Verification (CAV), 2001*. 2001.

[Swe02] Latanya Sweeney. "k-anonymity : A model for protecting privacy". In : *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), p. 557–570.

[TA15] Vinh-Thong Ta et Thibaud Antignac. "Privacy by Design : On the Conformance Between Protocols and Architectures". English. In : *Foundations and Practice of Security*. Sous la dir. de Frédéric Cuppens, Joaquin Garcia-Alfaro, Nur Zincir Heywood et Philip W. L. Fong. T. 8930. Lecture Notes in Computer Science. Springer International Publishing, 2015, p. 65–81.

[Tro+07] C. Troncoso, G. Danezis, E. Kosta et B. Preneel. "PriPAYD : privacy friendly pay-as-you-drive insurance". In : *Proc. of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007*. Sous la dir. de Peng Ning et Ting Yu. ACM. 2007, p. 99–107.

[WB90] Samuel D Warren et Louis D Brandeis. "The right to privacy". In : *Harvard law review* (1890), p. 193–220.

[Wes70] Alan F Westin. "Privacy and freedom. 1967". In : *Atheneum, New York* (1970).

[Yao82] Andrew C. Yao. "Protocols for secure computations". In : *Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on*. 1982, p. 160 –164.

[YLA04] Ting Yu, Ninghui Li et Annie I. Antón. "A Formal Semantics for P3P". In : *Proceedings of the 2004 Workshop on Secure Web Service*. SWS '04. New York, NY, USA : ACM, 2004, p. 1–8.

[ÁC11] Gergely Ács et Claude Castelluccia. "I Have a DREAM ! (DiffeRentially privatE smArt Metering)". In : *Information Hiding*. Sous la dir. de Tomáš Filler, Tomáš Pevný, Scott Craver et Andrew Ker. T. 6958. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, p. 118–132.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

FOLIO ADMINISTRATIF

THÈSE SOUTENUE DEVANT L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

NOM : Antignac — DATE de SOUTENANCE : 25 février 2015

Prénoms : Thibaud

TITRE : Méthodes formelles pour le respect de la vie privée par construction

NATURE : Doctorat — Numéro d'ordre : 2015ISAL0016

Ecole doctorale : Informatique et Mathématiques (InfoMaths — EDA 512)

Spécialité : Informatique

RESUME :
Le respect de la vie privée par construction est de plus en plus mentionné comme une étape essentielle vers une meilleure protection de la vie privée. Les nouvelles technologies de l'information et de la communication donnent naissance à de nouveaux modèles d'affaires et de services. Ces services reposent souvent sur l'exploitation de données personnelles à des fins de personnalisation. Alors que les exigences de respect de la vie privée est de plus en plus sous tension, il apparaît que les technologies elles-mêmes devraient être utilisées pour proposer des solutions davantage satisfaisantes. Les technologies améliorant le respect de la vie privée ont fait l'objet de recherches approfondies et diverses techniques ont été développées telles que des anonymiseurs ou des mécanismes de chiffrement évolués.

Cependant, le respect de la vie privée par construction va plus loin que les technologies améliorant simplement son respect. En effet, les exigences en terme de protection des données à caractère personnel doivent être prises en compte au plus tôt lors du développement d'un système car elles peuvent avoir un impact important sur l'ensemble de l'architecture de la solution. Cette approche peut donc être résumée comme « prévenir plutôt que guérir ».

Des principes généraux ont été proposés pour définir des critères réglementaires de respect de la vie privée. Ils impliquent des notions telles que la minimisation des données, le contrôle par le sujet des données personnelles, la transparence des traitements ou encore la redevabilité. Ces principes ne sont cependant pas suffisamment précis pour être directement traduits en fonctionnalités techniques. De plus, aucune méthode n'a été proposée jusqu'ici pour aider à la conception et à la vérification de systèmes respectueux de la vie privée.

Cette thèse propose une démarche de spécification, de conception et de vérification au niveau architectural. Cette démarche aide les concepteurs à explorer l'espace de conception d'un système de manière systématique. Elle est complétée par un cadre formel prenant en compte les exigences de confidentialité et d'intégrité des données. Enfin, un outil d'aide à la conception permet aux concepteurs non-experts de vérifier formellement les architectures. Une étude de cas illustre l'ensemble de la démarche et montre comment ces différentes contributions se complètent pour être utilisées en pratique.

MOTS-CLÉS : respect de la vie privée, méthodes formelles, vérification, logiques, génie logiciel, conception, architecture

Laboratoire (s) de recherche : Centre of Innovation in Telecommunications and Integration of service (CITI lab)
Inria, Université de Lyon/INSA de Lyon

Directeur de thèse :	Daniel Le Métayer	Directeur de recherche	Inria
Président de jury :	Fabrice Valois	Professeur des universités	INSA de Lyon

Composition du jury :

Rapporteurs	Sébastien Gambs	Chaire de recherche, HDR	Université de Rennes 1, Inria
	Gerardo Schneider	Professeur	Université de Göteborg
Examinateurs	Antonio Kung	Directeur technique	Trialog
	Patricia Serrano Alvarado	Maître de conférences	Université de Nantes
Président	Fabrice Valois	Professeur des universités	INSA de Lyon
Directeur	Daniel Le Métayer	Directeur de recherche	Inria